PERFORMANCE EVALUATION OF MULTI MACHINE VIRTUAL ENVIRONMENTS

Stefan Appel, Ilia Petrov, Alejandro Buchmann
TU Darmstadt
Databases and Distributed Systems Group
lastname@dvs.tu-darmstadt.de

Abstract Virtualization is already a common technology used in data centers as well as on desktop computers; it gains additional momentum with the spread of cloud computing. In this paper we analyze the performance and behavior of virtual resources in multi virtual machine scenarios running the same workload. We evaluate the performance in terms of CPU, memory and disk IO throughput. In doing so we vary the number of simultaneously running virtual machines to evaluate the fairness of resource sharing and the overhead due to virtualization, as well as the mutual interference among the virtual machines.

Introduction

As web applications gain popularity the number of users grows steadily. Thus, to fulfill the increasing resource demands, the number of servers per application increases whereas virtualization technologies can be applied to simplify management of servers while utilizing consolidation effects. An additional layer of abstraction between hardware and application is added allowing virtual machines to be created or removed easily and thus changing resource demands can be addressed quickly. Virtualization as such is a well-established technology designed in the 1970s by IBM for its mainframe facilities [Creasy, 1981]. Today desktop machines, low-end and mid-range servers are equipped with enough computational power to run several systems in parallel. There are many compelling use cases ranging from running multiple OS in parallel for development and testing purposes through server consolidation and green IT to cloud computing and HPC approaches. Although the majority of the existing research focuses on single virtual machine (VM) scenarios or targets multiple VMs executing different workloads, we claim that running multiple VMs with identical or very similar loads is a common case. Consider for example the following areas: multiple instances of hosted web applications, web server hosting, Data GRID applications and clustered web databases, big data. We argue that such workloads represent an equally compelling case, hence it is interesting to perform further research in this direction.

Conceptually speaking VMs are provided with virtual resources which are mapped onto the physical resources of the underlying hardware by the hypervisor. The efficiency of the mapping characterizes the quality of the hypervisor. For some resources the sum of the virtual resources can exceed the sum of the physical resources, hence virtual resources are shared or multiplexed. The question arises whether simultaneously running VMs with similar loads behave similarly or whether certain VMs are preferred? One of our goals is to study the behavior of virtual resources and draw conclusions about the resource sharing in virtualized environments; thus we especially emphasize possible interferences among the VMs. The study provided in the present paper encompasses the following resources: CPU, memory, disk IO.

The contributions of the paper are: (i) We study the behavior of virtual resources such as CPU, memory and disk IO based on microbenchmarks. It becomes clearly evident that CPU and memory are well suited for virtualization while for IO some interesting effects are reported. (ii) We study the scalability of virtual resources. We design experiments to study the scalability in the three areas where virtual resources are less than, equal to, or greater than the physical resources. (iii) Last but not least we evaluate the performance penalty incurred through the interference of multiple concurrent VMs running similar loads.

1. Related Work

There are different hypervisors for different processor and system architectures as well as for different operating systems (for Intel x86 VMWare and XEN, for SUN UltraSPARC LDOMS) [Smith and Nair, 2005]. Several existing studies characterize the IO behavior [Ahmad et al., 2003],[Ahmad, 2007]. Multiple studies exist on the CPU behavior, e.g. [Barham et al., 2003]. In addition there are two standard virtualization benchmarks VMMark [VMware, 2009], and the upcoming SPECvirt. What all of the above studies have in common is: they concentrate on multiple but dissimilar workloads. Some even consider only single VMs. The goal and original contribution of the present paper is to study the behavior of virtual resources when multiple concurrently running VMs execute a similar load.

2. Experimental Setup

For the evaluation of CPU, memory and IO behavior in virtualized systems we use an IBM x3850 Server with 4 Intel Dual-Core Xeon 7150N 3.5GHz CPUs, 16GB RAM and 6 10k RPM SAS hard disks configured as RAID 10. As virtualization software we select the product of a major vendor; the host operating system is a Debian/GNU Linux with a kernel version 2.6.18. We use OpenJDK 1.6.00-b11 as Java Runtime Environment (JRE). All VMs have the vendor-provided tools installed, allowing an optimal communication between hypervisor and VM. Ubuntu 8.04, kernel 2.6.24-server is used as guest operating system. The hardware configurations of the VMs are specified for every experiment respectively.

3. CPU Performance

The SPECjvm2008 benchmark suite [SPEC, 2008] is used to evaluate the CPU performance.[1] Java workloads represent standard CPU footprints for many business critical applications. SPECjvm2008 implements a CPU intensive workload utilizing multiple cores. Hence it provides a good basis for CPU evaluation in virtual environments. SPECjvm2008 contains twelve independent benchmarks; for a compliant run each of these benchmarks is executed for a certain period of time with a defined workload. After a run the number of completed iterations determines the final score for each benchmark; the score is reported in operations per minute (ops/m) whereas one operation equals one iteration of one of the benchmarks. The final score is then calculated based upon the single benchmark scores.

3.1 Configuration and Experiments

For the experiments each VM is configured with two virtual CPUs (vCPU) and 1024MB of RAM. SPECjvm2008 is executed simultaneously on a different number of VMs. The maximum heap size of the JVMs is set to 512MB so that still sufficient resources are available for the operating system running in the VMs. Five experiments are conducted using different test setups; for each experiment the number of VMs running in parallel is increased to force the system into heavy resource sharing mode.

Within each VM SPECjvm2008 is executed with identical parameters and it is started synchronously on all VMs.

3.2 Results

Table 1 shows the SPECjvm2008 scores for all five conducted experiments. Our starting point is a single VM experiment. Since there are sufficient resources on the host system the score of this experiment can be considered the maximum achievable and is therefore regarded as a reference point. We conduct subsequent experiments varying the number of simultaneously running VMs from two up to seven, whereas with four VMs and two vCPUs per VM the CPU capacity of the host system is reached (virtual CPU $\approx$ physical CPU). Hence adding a fifth VM requires CPU sharing. We observe that the SPECjvm2008 score remains constant for one to three parallel running VMs.

Table 1. Average SPECjvm2008 Scores, Parallel Execution in Virtual Machines

	Number of Virtual Machines				
	1	**4**	**5**	**6**	**7**
Average SPECjvm2008 Score	14.770	14.060	11.786	9.707	8.094
Standard Deviation	-	0.121	0.084	0.110	0.100
Accumulated Score	14.770	56.240	58.930	58.240	56.660

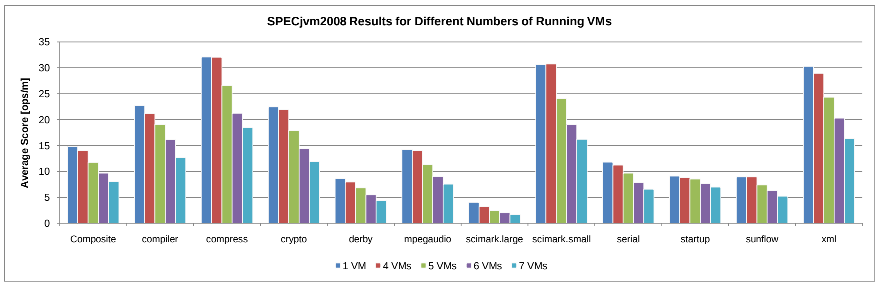

Figure 1. Average SPECjvm2008 Score, Overall and per Benchmark

With four VMs the capacity of the host system is reached. Thus CPU is becoming a bottleneck resulting in lower SPECjvm2008 scores. Increasing the number of VMs further causes more CPU sharing which leads to significantly lower SPECjvm2008 scores per VM.

A detailed result overview is shown in Figure 1; the total SPECjvm2008 score (Composite) as well as the scores for the separate benchmarks are shown. When comparing the 1 VM and 4 VMs scenario an interesting effect can be observed: while for some benchmarks, like *compress* or *sunflow*, no decrease in score occurred, for other benchmarks, like *compiler* or *xml*, the score decreases significantly. An explanation for this behavior is the CPU usage of SPECjvm2008: not all of the benchmarks contained in SPECjvm2008 use both available vCPUs all the time as it can be seen in Figure 2. Thus no sharing of resources is necessary for some benchmarks and the score does not decrease (e.g. for *compress* and *sunflow*).

3.3 Conclusion

Table 1 clearly shows that the standard deviations remain below 1.2%. Hence we can conclude that CPU time is distributed fairly amongst all VMs; otherwise higher standard deviations would occur. Although overhead due to virtualization and CPU sharing increases with an increasing number of VMs the accumulated score decreases only slowly and the system handles multiplexing well. Table 1 shows the accumulated score, which increases for one to five VMs, due to available CPU capacity, and decreases due to overhead effects for

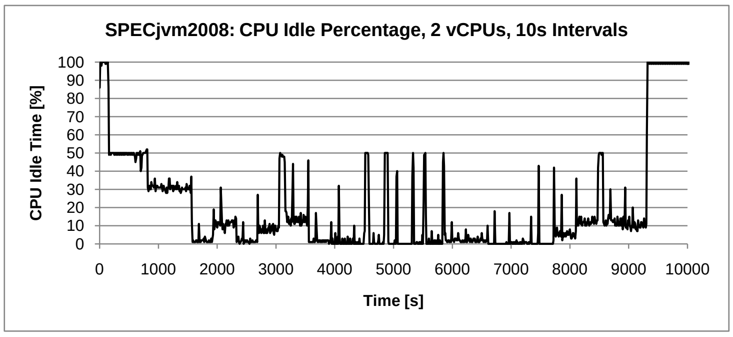

Figure 2. CPU Idle Time during SPECjvm2008 Run

five to seven VMs (system overloaded). Amongst others the following effects contribute to the overall overhead: (i) CACHE MISSES: Caches have a major impact on the performance of modern systems. Since with every context switch to another VM a different part of the memory is addressed many cache misses occur when giving CPU time to the next VM. (ii) CONTEXT SWITCHING: The system has to perform scheduling and context switching to ensure that CPU time is shared fairly and no VM is preferred over another. Ultimately the overhead for these operations increases with an increasing number of VMs. (iii) UNDERLYING OPERATING SYSTEM: Finally the underlying operating system requires resources as well. Although these cannot be used by the VMs, they cannot be ignored.

4. Memory Throughput

Besides CPU, memory is another important resource which has big influence on the performance of a VM. Two memory aspects are of interest: (a) the throughput in case of multiple VMs performing memory operations; (b) the systems behavior if the memory assigned to the VMs exceeds the physically available memory. The latter scenario becomes critical if VMs start making heavy use of memory causing swapping which entails significant performance degradation. The following evaluation concentrates on scenarios where sufficient memory is available whereas the main aspect is the reachable throughput of each VM when all VMs execute memory intensive operations.

4.1 The RAMSMP Benchmark

We perform experiments using two memory microbenchmarks RAMSPEED and RAMSMP ([Hollander, 2008]), however we concentrate on the RAMSMP results. The difference between RAMSPEED and RAMSMP is that the former only utilizes one CPU core whereas the latter can be set up to start multi-

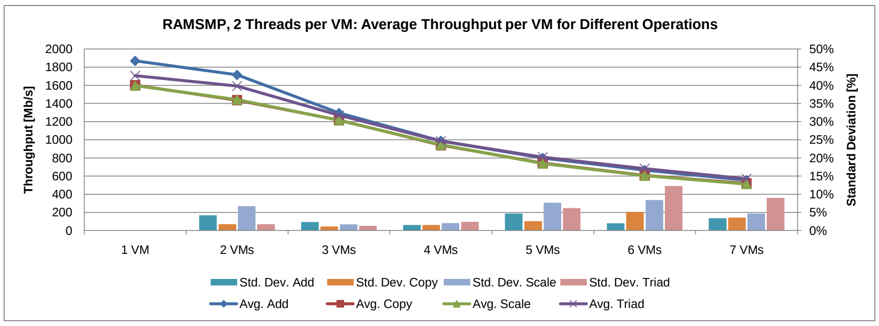

Figure 3. RAMSMP - Average Throughput

ple threads in order to allow testing multi-core systems. The two benchmarks measure the memory throughput by executing four different types of operations [Hollander, 2008]: (i) COPY: Transfers data from one memory location to another, i. e. $A = B$. (ii) SCALE: Modifies the data before writing by multiplying with a certain constant value, i. e. $A = m \cdot B$. (iii) ADD: Reads data from the first memory location, then reads from the second, adds them up and writes the result to the third place, i.e. $A = B + C$. (iv) TRIAD: Reads data from the first memory location, scales it, then adds data from the second one and writes to the third place, i.e. $A = m \cdot B + C$.

The measured throughput as well as the used operations are comparable to results obtained using the STREAM benchmark ([McCalpin, 1995]), however RAMSPEED and RAMSMP provide better capabilities for automated testing and aggregation of results.

4.2 Configuration and experiments

We use VMs configured with 2 vCPUs and 2048MB RAM each. RAMSMP is used in batch mode which allows automated execution of several benchmark runs and reports averaged results. We vary the number of concurrently running VMs while performing five consecutive RAMSMP runs. During each run the benchmarks write a total of 8GB of data; RAMSMP uses one thread on each vCPU. The benchmarks are started simultaneously in all VMs.

4.3 Results

Figure 3 shows the average throughput as well as the standard deviation per VM. With four and more VMs all physical CPU cores are fully utilized and the standard deviations increase. While the standard deviation is below 5% for all but one test in the two to four VMs scenarios, with five to seven VMs the standard deviation exceeds the 5% limit six times.

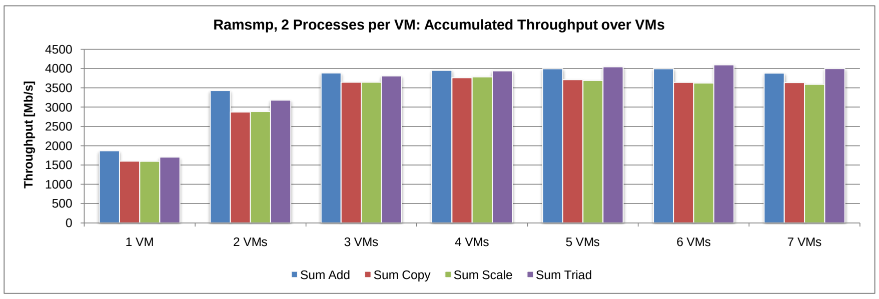

Figure 4. RAMSMP - Accumulated Throughput

Let us consider the accumulated throughput over all VMs for RAMSMP. While the maximum throughput for RAMSPEED is reached with six VMs, it is reached with three VMs for RAMSMP; both scenarios (3 VMs RAMSMP, 6 VMs RAMSPEED) are identical in terms of utilized CPU cores. Thus it can be concluded that the system scales independent from the actual number of running VMs - the memory throughput is only influenced by the number of utilized vCPUs. Further the maximum throughput for RAMSMP and RAMSPEED is equal whereas a slight decrease of throughput can be observed for RAMSMP with 5 and more VMs running. This can be explained with overhead effects occurring due to resource sharing as mentioned in Section 3.2; especially caching effects are of interest here since these significantly influence memory performance.

4.4 Conclusion

The conducted experiments show that memory bandwidth is shared fairly among all concurrently running VMs. Moreover the maximum throughput of the whole system can only be reached when many CPUs are utilized to take advantage of all the caching effects; the highest overall throughput for our scenario can be observed with three running VMs. With more VMs, the throughput deteriorates slightly; but even with 14 utilized vCPUs the accumulated throughput is still close to the maximum.

5. Disk IO Throughput

The IO behavior in virtual environments is a critical factor for virtualizing data-intensive applications and systems such as database systems. In the present section we characterize the filesystem IO performance based on several different benchmarks and evaluate how IO bandwidth is divided among multiple VMs.

Table 2. Average Throughput (in KB/s) measured with Bonnie++, 1 VM, 4 Runs

	Sequential Output (write)			**Sequential Input (read)**	
	Write (Character)	*Write (Block)*	*Rewrite (Block)*	*Character*	*Block*
Throughput	32560.75	73055.00	62697.00	43035.25	203317.25
SD	2539.01	1852.36	1712.27	1543.06	4560.33

Table 3. Average Throughput (in KB/s) measured with Iozone, 1 VM, 6 Runs

	Write		**Re-Write**		**Read**		**Random Read**	
Blocksize	64kB	128kB	64kB	128kB	64kB	128kB	64kB	128kB
Throughput	79966	82168	80068	79501	238010	235990	113467	122939
SD	3102.67	1074.00	2276.11	1635.85	7157.98	8642.38	3025.16	3991.70

5.1 Benchmarks

For IO evaluation we choose *Bonnie++* [Coker, 2009] as well as *Iozone* [Norcott and Capps, 2006]. Bonnie++ is a simple benchmark which performs sequential read and write operations using *putc(), getc()* (character-wise) and *write(), read()* (block-wise). Iozone performs many more operations; besides normal read and write it performs re-read and re-write operations in order to make use of caches, like those available on RAID controllers. Depending on the configuration, Iozone does perform various read and write operations for different file sizes and block sizes so that it is possible to get insight into the IO system characteristics.

5.2 Configuration

The VMs are configured with 2 vCPUs each as well as 1024MB of RAM. Bonnie++ as well as Iozone are configured to create, read and write a 2GB file; this is twice the RAM size in order to avoid buffering the whole file in memory.

5.3 Results

Consistency of Results. We perform an initial reference experiment with one VM running both benchmarks successively. Table 2 shows the average throughput as well as the standard deviations of four consecutive Bonnie++ runs. The highest standard deviation is 7.8% which is a good value for an IO benchmark. Thus Bonnie++ results can be seen as reproducible. Table 3 shows the average throughput and standard deviation for six consecutive Iozone runs. Iozone performs 13 different tests using nine different block sizes for each test; for a better comparability only results for small block sizes and tests similar to Bonnie++ are shown. The overall standard deviation of the Iozone results is below 5%. Thus these can also be seen as reproducible and reliable. Throughputs for larger blocks are to some extent higher than for small blocks however the standard deviations remain low.

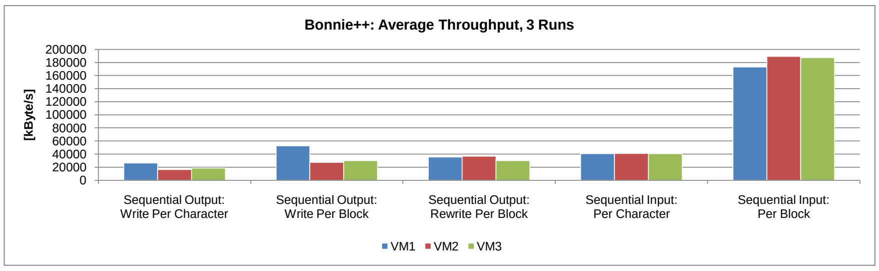

Figure 5. Bonnie++ - Average Throughput per VM, 3 Runs, 3 VMs

The Bonnie++ read and write throughput when writing character wise is significantly lower than the values reported for the corresponding block wise operations (Table 2). Furthermore the throughput values for Iozone (Table 3) exceed the corresponding values reported by Bonnie++. The Iozone results for the smallest block size are twice as high for writing (Bonnie++, per-character) and five times as high for reading (Bonnie++, per-character). A feasible explanation is that Iozone does not perform any Per-Character-Operations at all. Bonnie++ results for reading and writing block-wise are closer to the Iozone results but still differ as much as 25% (for Re-Write). Although each benchmark generates consistent and reproducible results the comparability between them is not provided. Especially problematic is the fact that Bonnie++ does not explicitly inform about the used block size.

Resource Sharing. The first test setup consists of three VMs running in parallel. The server has sufficient CPU and memory resources for three VMs so only the IO resources are shared. All in all three Bonnie++ and two Iozone runs are conducted with this test setup whereas the benchmarks are started simultaneously in all VMs. In the ideal case the IO bandwidth would be distributed evenly among all VMs; this would result in a low standard deviation for the average throughput. The conducted experiments do not exhibit this; for all experiments high standard deviations occurred.

Figure 5 shows the average throughput per VM measured with Bonnie++; in addition Figure 6 shows the standard deviations for these experiments. When only comparing the average throughput per VM it seems the bandwidth is distributed evenly amongst the VMs, just during "Sequential Output: Write Per Block" VM 1 reaches significantly higher throughput than other VMs. But when taking the standard deviations into consideration it can be seen that there are high fluctuations; this shows that although the bandwidth is distributed evenly on average, it is not distributed evenly during each run.

Figures 7 and 8 show the measurement results using Iozone. As with Bonnie++ the average throughput is relatively equal for all VMs, but also high

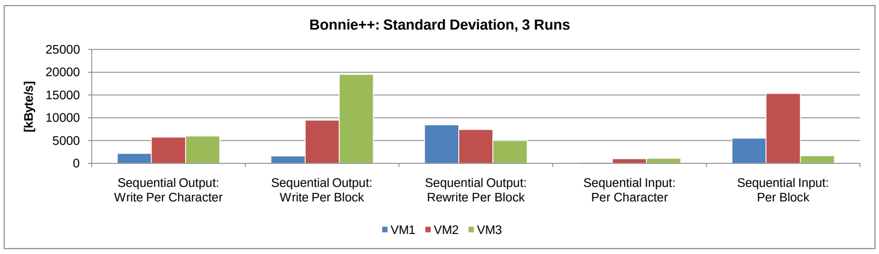

Figure 6. Bonnie++ - Standard Deviation per VM, 3 Runs, 3 VMs

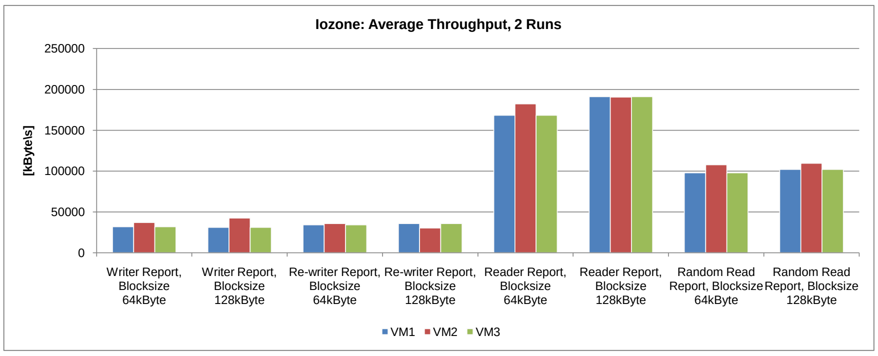

Figure 7. Iozone - Average Throughput per VM, 2 Runs, 3 VMs

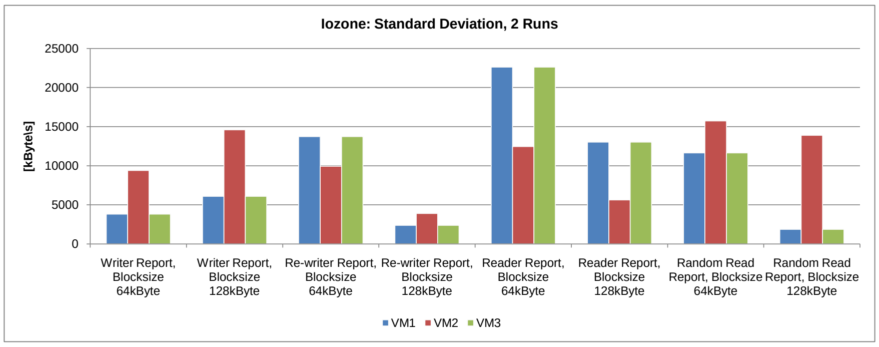

Figure 8. Iozone - Standard Deviations, 2 Runs, 3 VMs

standard deviations are observed, almost all exceeding 10%. Thus the Iozone results confirm the Bonnie++ results and show that even in case of sufficient CPU resources the distribution of IO bandwidth is not reliably predictable. The Iozone results for larger block sizes are similar: high standard deviations occur. Thus the observed behavior is not specific for small blocks.

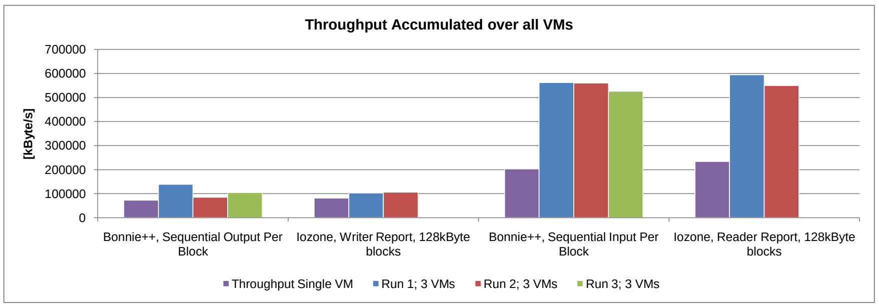

Figure 9. Iozone & Bonnie++ - Accumulated Throughput over 3 VMs

Although the measurement results show high fluctuations of throughput an interesting behavior can be noticed for both benchmarks: the accumulated throughput over all three running VMs exceeds the throughput a single VM is able to reach (cp. Tables 2 and 3). This is true for each of the three Bonnie++ runs and for both of the Iozone runs. Figure 9 shows the accumulated throughput for some of the tests; the leftmost bars in each group denote the measured throughputs of a single VM scenario. While the effect is small for write operations, for read operations a major difference is noticeable. A possible explanation are caching effects within the VMs as well as on the host system. While the increased read throughput can be explained by the increased number of VMs (each VM can buffer data), explaining the increase of throughput for write operations is more difficult. One possibility is the serialization of write requests by the hypervisor so that larger amounts of data can be written to the disks sequentially which results in higher throughput. Further the caching of write operations can cause this behavior as well.

Besides a test setup with three VMs another setup with five VMs is evaluated. The configuration of the VMs remains unchanged but with the two additional machines the number of available CPU cores is not sufficient anymore; overall ten vCPUs are assigned to VMs. The five VM experimental results show tendencies similar to those of the three VM scenario.

5.4 Conclusion

At first it is shown, that Bonnie++ and Iozone produce reliable and reproducible results; thus both benchmarks are used to evaluate IO performance in VMs. Two different test setups are used: one with three and one with five VMs running in parallel. All results show that IO performance in virtualized environments is difficult to predict. Although bandwidth is distributed evenly on average, high standard deviations occur, showing that the throughput of each VM differs from run to run. However the results show that despite high fluctuations one interesting effect can be observed: the accumulated throughput over all running VMs exceeds the throughput of one VM; especially for writing this is not expected.

6. Conclusions

The main goal of the present paper is to study the behavior of virtual resources in multi VM environments running similar loads. We also aim at studying whether resources are shared fairly among VMs in such setups. We analyze CPU, memory and disk IO performance. CPU performance is evaluated using the SPECjvm2008 benchmark showing that CPU time is distributed evenly among the VMs. Further the overhead due to virtualization is analyzed and it is shown that it increases with an increasing number of VMs whereas the system handles an overload well. It can be observed that memory bandwidth is shared evenly among the running VMs. The highest throughput (accumulated over VMs) can be reached with a 75% utilized system, which is probably due to optimal utilization of all caches. The last evaluated resource is the disk IO system. In contrast to the CPU and memory, disk IO bandwidth is not distributed evenly all the time. Although the average throughput over several runs does not differ heavily among VMs, high standard deviations indicate that the sharing of IO bandwidth is problematic. Finally it can be observed that the accumulated throughput over all VMs exceeds the throughput of one running VM.

Notes

1. SPECjvm2008 is a trademark of the Standard Performance Evaluation Corporation (SPEC). The results or findings in this publication have not been reviewed or accepted by SPEC, therefore neither comparison nor performance inference can be made against any published SPEC result. The official web site for SPECjvm2008 is located at http://www.spec.org/jvm2008

References

[Ahmad, 2007] Ahmad, I. (2007). Easy and efficient disk i/o workload characterization in vmware esx server. In *Proc. of the IISWC '07*, pages 149–158.

[Ahmad et al., 2003] Ahmad, I., Anderson, J., Holler, A., Kambo, R., and Makhija, V. (2003). An analysis of disk performance in vmware esx server virtual machines. In *Workload Characterization, 2003. WWC-6. 2003 IEEE International Workshop on*, pages 65–76.

[Barham et al., 2003] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. (2003). Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5):164–177.

[Coker, 2009] Coker, R. (2009). bonnie++. `http://freshmeat.net/projects/bonnie/`.

[Creasy, 1981] Creasy, R. J. (1981). The origin of the vm/370 time-sharing system. *IBM Journal of Research and Development*, 25(5):483–490.

[Hollander, 2008] Hollander, R. M. (2008). Ramspeed, a cache and memory benchmark. `http://www.alasir.com/software/ramspeed`.

[McCalpin, 1995] McCalpin, J. D. (1995). Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pages 19–25.

[Norcott and Capps, 2006] Norcott, W. D. and Capps, D. (2006). Iozone filesystem benchmark. `http://www.iozone.org/`.

[Smith and Nair, 2005] Smith, J. and Nair, R. (2005). *Virtual Machines: Versatile Platforms for Systems and Processes*. Morgan Kaufmann Inc.

[SPEC, 2008] SPEC (2008). Specjvm2008 (java virtual machine benchmark). `http://spec.org/jvm2008/`.

[VMware, 2009] VMware (2009). Vmware: Measure virtualization performance with the industry's first benchmark. `http://www.vmware.com/products/vmmark/`.