SPEK: A Storage Performance Evaluation Kernel Module for Block Level Storage Systems

Ming Zhang*and Qing Yang
Electrical and Computer Engineering
University of Rhode Island
Kingston, RI 02881 USA
{mingz, qyang}@ele.uri.edui

Xubin He
Electrical and Computer Engineering
Tennessee Technological University
Cookeville, TN 38505, USA
hexb@tntech.edu

Abstract

In this paper we introduce SPEK (Storage Performance Evaluation Kernel module), a benchmarking tool for measuring and characterizing raw performance of data storage systems at block level. It can be used for both DAS (Direct Attached Storage) and block level networked storage systems. Each SPEK tool consists of a controller, several workers, and one or more probers. Each worker is a kernel module generating I/O requests to lower level SCSI layer directly. Compared to traditional file system and disk I/O benchmarking tools, SPEK is highly accurate and efficient since it runs at kernel level and eliminates file system overheads. It is specially suitable for accurately measuring raw performance of data storages at block level without influence of file system cache or buffer cache. Using SPEK, a user can easily simulate realistic workloads and produce detailed profiling data for networked storage as well as DAS. We have built a prototype on Linux and our experiments have demonstrated its accuracy and efficiency in measuring block level storage systems.

*Corresponding Author. Tel:1(401)874-2293, Fax:1(401)782-6422

Key words: *Performance tool, benchmarking, data storage, block level access, networked storage*

1. Introduction

Data storage has evolved from "secondary peripheral" to a primary and central part of a computing system because of importance of data. There are generally two types of storage systems, DAS (Direct Attached Storage) such as IDE/SCSI disks and disk arrays [1], and networked storage such as NAS (Network Attached Storage) [2, 3] and SAN (Storage Area Network) [4]. To satisfy ever-growing demands for data storage systems with high performance, reliability, and availability, new architectures, standards, and products emerge rapidly [1, 4, 5, 6]. In order to quantitatively evaluate various storage systems, accurate and efficient benchmark tools are needed. Such benchmark tools helps researchers, designers, and users to measure, characterize, and compare different storage systems.

Under many circumstances, the performance of a storage available to users is the performance result gotten from file systems. Since it is influenced by many other factors such as file system cache, data organization, and buffer cache be-

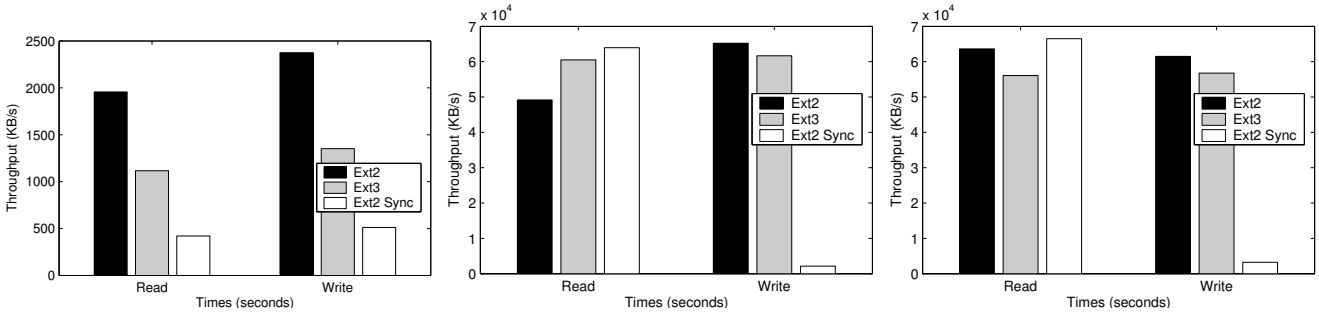


Figure 1. Measured throughput of PostMark (left), IoZone (middle), and Bonnie++ (right). Although using the same hardware, measured performance change dramatically with changes of file system options

sides the storage system, it cannot represent the real performance of the evaluated system. Only the raw performance, or block level performance, can be used to accurately compare different storages. Some applications, such as database, can utilize this raw performance directly and thus it is important to know how much they can get from a storage system. And raw performance values are also important for file system and OS designers to know how much raw performance they can exploit and how much optimization they make.

Existing benchmark tools such as PostMark [7], IoZone [8], Bonnie++ [9], and IoMeter [10] are widely used to measure various storage systems. PostMark, IoZone, and Bonnie++ run at file system level and therefore are mainly for characterizing file system performance. While they can be used for both DAS and networked storage, they are not suitable for measuring raw performance or block level performance. Measurement results using these benchmark tools are often skewed because of file system cache. Figure 1 shows experimental performance measurements of a same SCSI disk under different file systems (Ext2, Ext3, and Ext2 with Sync flag) using PostMark, IoZone, and Bonnie++, respectively. It is clearly shown that although our measured storage target is exactly same, these benchmark tools produce completely different performance results because of different file systems. Such deviations can be attributed to effects of file system cache as well as different characteristics of file systems [11]. Therefore, these kinds of benchmarks cannot provide accurate assessment of block level storage systems. While IoMeter can run below file systems, its measured performance on Linux fluctuates dramatically due to the effects of buffer cache at Linux block device layer. As will be evidenced in Section 3, measured results using IoMeter can differ from actual performance by as much as 600%. An important issue is that although these benchmark tools are not accurate when measuring block level performance, they are popularly used to measure raw performance in real life.

Besides the accuracy problem of existing benchmark tools, there is also an efficiency issue. Because these benchmarks run in user space above file system level, there are excessive system calls and context switches resulting in large amounts of overhead. This efficiency problem is more pronounced when measuring high performance networked stor-

age systems because intensity of traffic generated by these benchmarks is limited due to excessive system overheads. As a result, a large number of clients is needed to saturate a high performance block level networked storage.

In this paper, we propose a new benchmark tool called SPEK (Storage Performance Evaluation Kernel module), for measuring and characterizing data storage systems such as DAS and networked storages at block level. SPEK consists of a SPEK Controller, several SPEK Workers that generate I/O requests, and several SPEK Probers recording system status. Each SPEK Worker runs as a kernel module and sends all I/O requests to storage device drivers directly, bypassing file system cache and buffer cache. It is highly efficient since it runs at kernel space and minimizes overheads caused by system calls and context switching. It allows a user to configure workload parameters and define performance metrics of interest through an easy-to-use Java GUI interface. A set of utilities and scripts is also provided for users to automate testing process. Prototyping code has been developed on Linux as an open source tool under GPL license. Our experiments have shown that SPEK is highly accurate and highly efficient for measuring block level storage systems.

The paper is organized as follows. In Section 2, we describe the design of our SPEK in detail. Experimental validation in comparison with existing benchmarks is presented in Section 3. We briefly discuss existing I/O benchmark tools in Section 4 and conclude the paper in Section 5.

2. Structure and Behavior of SPEK

The overall structure of SPEK is shown in Figure 2. It consists of three main components, one SPEK Controller,

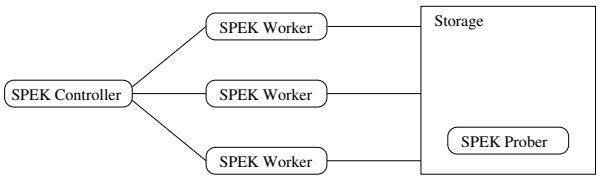


Figure 2. SPEK structure

several SPEK Workers, and one or more SPEK Probers. SPEK Controller resides on a controller machine which is used to coordinate SPEK Workers and Probers. It can start/stop SPEK Workers and Probers, send commands and receive responses from them. A Java GUI interface of a SPEK Controller allows a user to input configuration parameters such as workload characteristics and to view measured results. Each SPEK Controller also has a data analysis module to analyze measured data.

There is one SPEK Worker running on each testing client to generate storage requests via the low level device driver and record performance data. A SPEK Worker is a Linux kernel module running in kernel space. Each SPEK Worker has one main thread, one working thread, and one probe thread. The main thread is responsible for receiving instructions from SPEK Controller and controlling the working thread to execute the actual I/O operations. The working thread keeps sending requests to SCSI layer that are eventually sent to remote targets by lower level device driver. By using an event-driven architecture, SPEK is able to perform several outstanding SCSI requests concurrently, which is useful and necessary when testing SCSI tagged commands [12] feature and exploring the maximum throughput of a remote SCSI target. Many modern SCSI storages have the *command queue* feature that allows hosts to send several tagged commands and decide the specific execution

sequence based on their own scheduling policies to get maximum overall throughput. The probe thread records system status data periodically and reports to SPEK Controller once a test completes. On each target device, there is a SPEK Prober thread that records system status for post-processing. Currently, we have developed a SPEK Prober for Linux and plan to build SPEK Probers for other platforms. Its functionality is similar to probe thread in a SPEK Worker.

2.1. Configuring Workload Characteristics

SPEK workloads are generated by user configurable parameters similar to IoMeter. Each Worker generates workloads independently from other SPEK Workers allowing realistic networking environment to be simulated. Some of the parameters are:

- Block size. It is data block size of a storage request and is a multiple of sector size (512 Bytes). Currently we support up to 8 different block sizes in one test run and there is a frequency weight associated with each request block size. A sample workload may contain 10% of 8KB, 20% of 16KB, 30% of 32KB, and 40% of 64KB.
- Number of transactions. It controls how many transactions to be carried out in a test run. A transaction is defined as a block level read/write access.
- Ramp up count. It is a number used to bypass transient period of measurement process. Performance recording starts after number of requests finished exceeds this number.
- Burstiness is defined by the length of a bursty request and interval between two successive bursts. As a special case, when the interval is zero, SPEK sends requests continuously till all requests are finished without delay.
- Maximum outstanding request number. In order to test the SCSI tagged commands feature, the outstanding I/O request number is also configurable.
- Read/Write Ratio and sequential/random ratio. SPEK Worker generates random read/write requests based on these parameters. We use a random number generator that can generate numbers evenly distributing between two integers.
- Request Address Alignment. It defines how request address should align to.
- Report time interval. It defines the interval for a SPEK Worker to report performance data. Specially, if it is zero, a SPEK Worker only reports all data at the end of one test run.

2.2. Performance Metrics

SPEK reports mainly two performance values: throughput and response time. Throughput is represented in two forms: average I/O per second (IOPS) and average mega bytes per second (MBPS). Response time includes average response time and maximum response time. All average values we mention are mean values, while users can use raw data to get median value or other statistics easily. During each test run, SPEK collects data related to performance and system status. There are two options to record and transfer such data to SPEK Controller, periodically at run time or one time at the end of each run. Unlike many other benchmark tools that collect some statistical data and compute them on the fly, SPEK provides two options: (1) deferring computation/analysis while allowing more data to

be collected or (2) computing/analyzing on the fly. The former requires more memory space whereas it provides more detail data to analyze performance dynamics of measured targets and gives users more flexibility to process and analyze measured raw data. A user can play trade-offs between memory and flexibility when doing performance testing.

In addition to throughput and response time, SPEK records other profiling data such as CPU utilization, user time, system time, interrupts per second, and context switches per second. Furthermore, network related load status including receive/send packets per second and receive/send bytes per second, and memory load status such as free memory size, shared memory size, buffered memory size, swap size, swap exchange rate and so forth are also collected. All these system status data are recorded periodically with a user configurable interval.

3. Experimental Validation

In order to verify the promises of SPEK, we have carried out experiments to measure performance of DAS as well as networked storage using SPEK in comparison with existing benchmark tools. As mentioned in the introduction, most existing benchmark tools run at file system level with few exceptions such as IoMeter. We therefore compare our SPEK with IoMeter in terms of accuracy and efficiency.

3.1. Experiment Environment

Several PCs are used in our experiments. One acts as SPEK Controller, three act as test clients, and one as test target. All PCs are connected by an Intel NetStructure 470T Gigabit Switch via Intel Pro1000 Gigabit NICs. The detailed hardware configurations of these PCs are shown in Table 1. All PCs run Redhat Linux 7.3 with recompiled 2.4.18 kernel. Detail specifications of the measured SCSI disks are shown in Table 2.

	Test Client	Test Target
CPU	1 PIII 800MHZ	1 PIII 866MHZ
Memory	256M PC133	512M PC133
NIC	Intel Pro1000T	Intel Pro1000T
DISK		IBM and Seagate SCSI disks
Controller		Adaptec 39160

Table 1. Test machines configuration

In our first experiment, we measured the random write performance of Seagate SCSI disk in terms of IOPS with each request size being 16KB as shown in Figure 3. It is interesting to observe that throughputs produced by IoMeter fluctuate dramatically between 0 and 300 IOPS while those produced by SPEK are fairly consistent over time. The fluctuation of the throughputs produced by IoMeter results mainly from the buffer cache. Because of the existence of the buffer cache, throughputs are high at times. However, Linux flushes the buffer cache when large enough sequential blocks are accumulated, every 30 seconds, or when dirty data exceeds a threshold value. Since our workload is random write, it is very unlikely to accumulate large sequential blocks. Most of flushing is caused by timeout and excessive dirty data. During a flushing period, measured throughput approaches zero because the system is busy and not able to respond to normal I/O requests. This fact clearly indicates the limitation of IoMeter in accurately measuring disk I/O performance. Our SPEK module, on the other hand, produces accurate and stable throughput values over time since SPEK runs at lower layer and is not affected by buffer cache as shown in Figure 3.

The accuracy of SPEK is further evidenced by Figure 4

	Model	Interface	Capacity (GB)	Data Buffer (MB)	RPM	Latency (ms)	Sustained date rate (MB/s)	Seek time (ms)
IBM	DNES-309170	Ultra2 SCSI	9.1	2	7200	4.17	12.7 to 20.2	7.0
Seagate	ST318452LW	Ultra160 SCSI	18.4	8	15000	2.0	N/A	3.6/4.2

Table 2. SCSI disk parameters

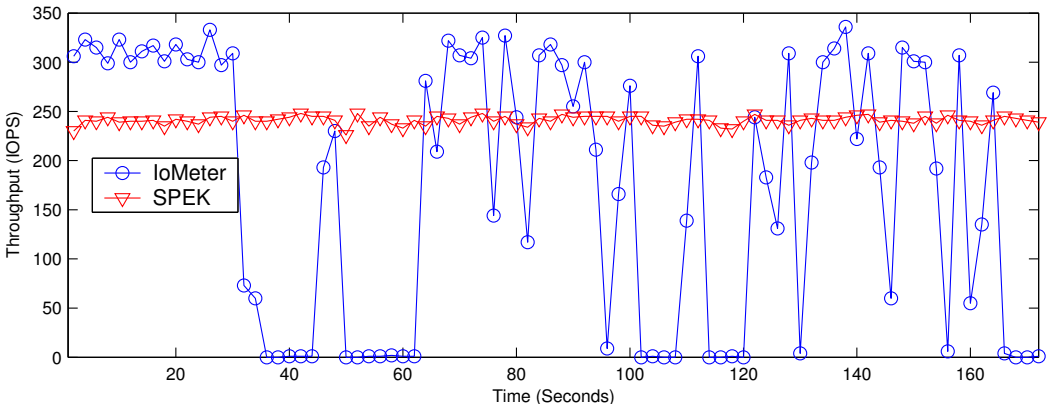
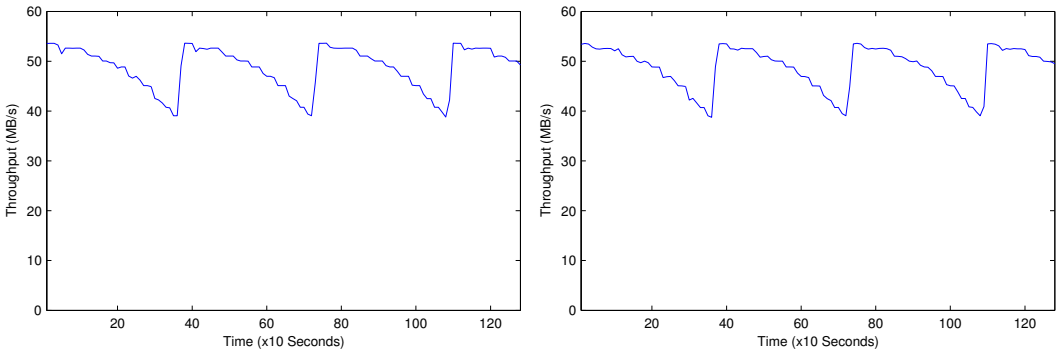


Figure 3. Measured throughputs for random write with block size being 16KB. The average IOPS of IoMeter is 175 while that of SPEK is 240. And the dynamic result of IoMeter fluctuates between 0 and 300 because of buffer cache effects while that of SPEK keeps consistent.



(a) Sequential read workload (b) Sequential write workload

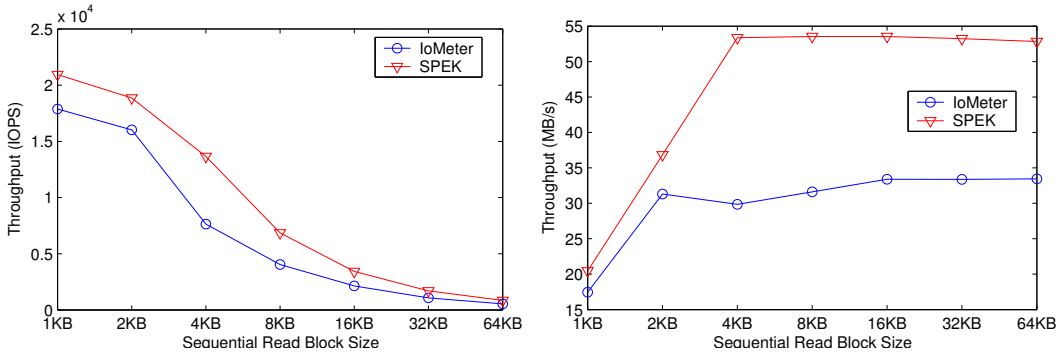
Figure 4. Measured throughput on the Seagate SCSI disk using SPEK. SPEK correctly captures the ZCAV scheme of the Seagate SCSI disk.

that shows throughputs of the Seagate disk under sequential read and sequential write workloads. In this figure, throughput changes periodically between 55MB/s and 39MB/s. We noticed that the total data accessed in each period is 18GB which is approximately the formatted disk size. With Zoned Constant Angular Velocity (ZCAV) scheme, a modern SCSI disk has more sectors on outer tracks than inner tracks. As a result, accessing sectors on outer tracks is faster than inner tracks giving rise to the periodic throughput change as shown in the figure.

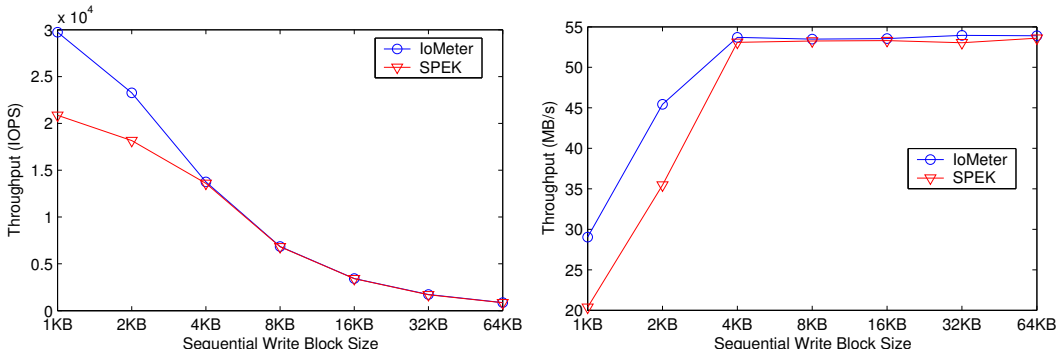
To provide a comprehensive comparison between IoMeter and SPEK, we carried out measurement experiments across other different types of workloads as shown in Figure 5. For sequential read workloads (Figure 5a), IoMeter achieves lower throughput than SPEK. In terms of MBPS, throughput of IoMeter saturates at about 33MB/s while SPEK saturates at about 53MB/s. The difference results mainly from the system overheads for managing the file system cache and buffer cache that do not provide any performance benefit because of sweeping data access without reuse. Note that for read operations, Linux system will copy data read from lower level to the file system cache for possible future reuse. For sequential write as shown in Figure 5b, IoMeter produces better throughputs than SPEK for small request sizes. This is because written data bypass file system cache, and the buffer cache collects small writes to form large sequential writes resulting in better write throughputs. As the request size increases, such difference diminishes. All these measured data clearly indicate that throughputs produced by IoMeter are strongly influenced by the file system cache and the buffer cache. They do not accurately represent the actual performance of the underlying disk storage. SPEK, on the other hand, accurately measures the raw performance of the block level storage devices.

In the case of random read workloads shown in Figure 5c, measured throughput by both IoMeter and SPEK are fairly close. The reason is that overheads of file system in this situation are negligible compared to tens of millisecond disk operations involving random seek, rotation latency, and transfer. Furthermore, the effect of file system cache is also negligible because we generated 200,000 random read requests uniformly distributed over 18GB space giving rise to approximately zero cache hit ratio. For random write workload as shown in Figure 5d, the results are consistent with those in Figure 3 for the same reasons explained previously. Note that Figure 5d shows the average throughput whereas Figure 3 shows the instant throughput measured at a particular time point. Similar results are observed when measuring IBM disk as shown in Figure 6. The throughput difference produced by the two benchmarks is as high as 600%.

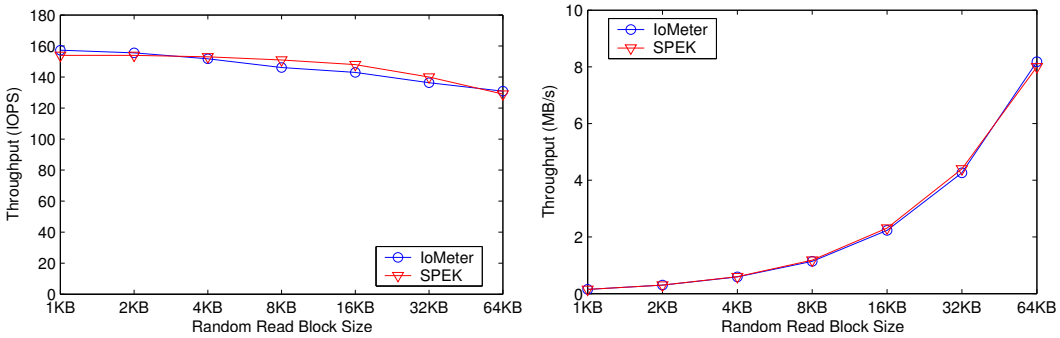
The target disks measured in above experiments are not very high performance disk storages. For example, an entry level RAID system such as Dell/EMC CX200 [13] has up to 25,000 cache IOPS and high-end IBM Shark F20 can have 11,000 IOPS even with 0% cache hit. An even faster SSD device like RamSan-210 can provide 100,000 random IOPS with one port. To measure such high performance storage systems, the advantages of our SPEK become more evident because file system overheads are no longer negligible as compared to high speed disk access times. To observe how SPEK and IoMeter perform in measuring such high performance storage systems, we carried out experiments on high speed storages. With the absence of real hardware of these expensive storages, we use a software simulator to simu-



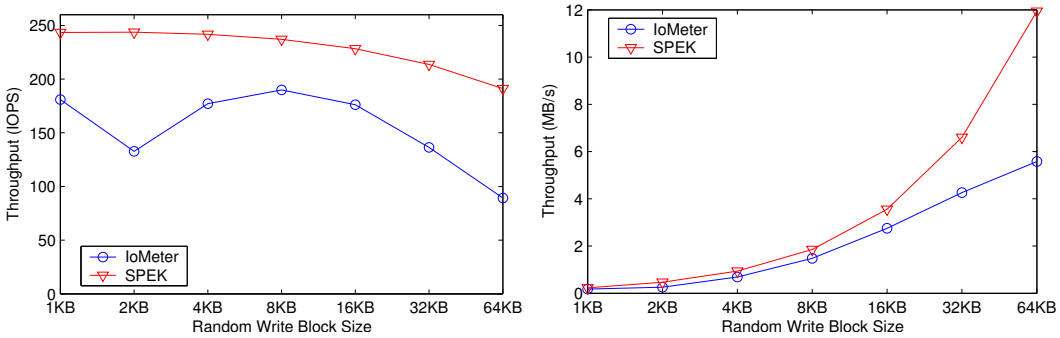
(a) Sequential read workload



(b) Sequential write workload

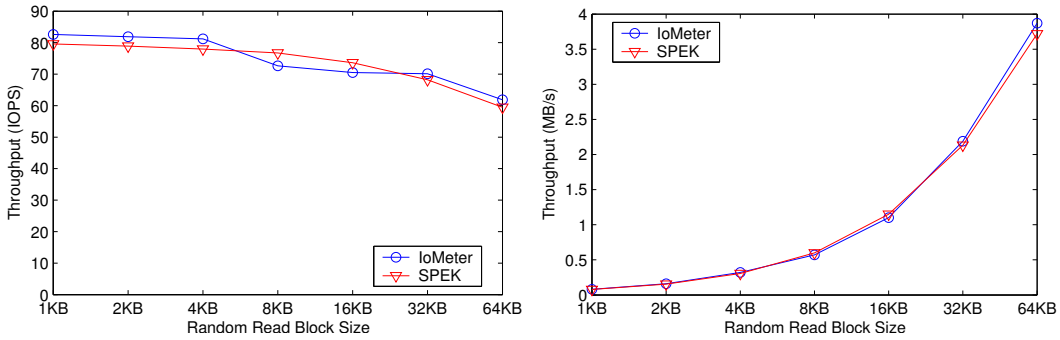


(c) Random read workload

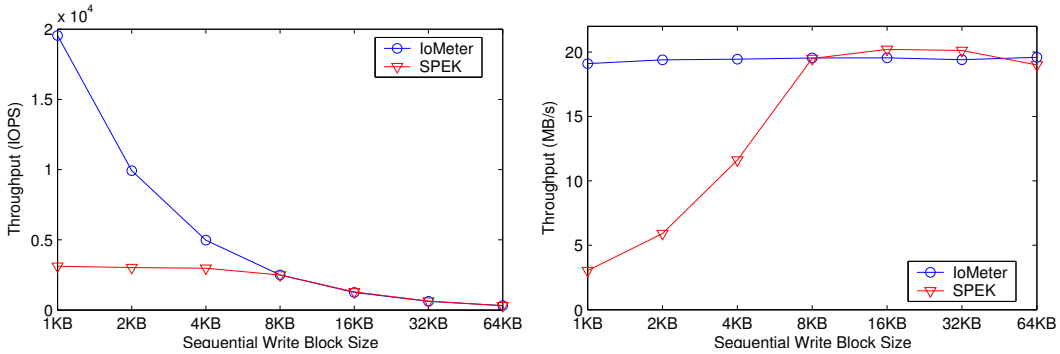


(d) Random write workload

Figure 5. Measurement results on Seagate disk with different workloads

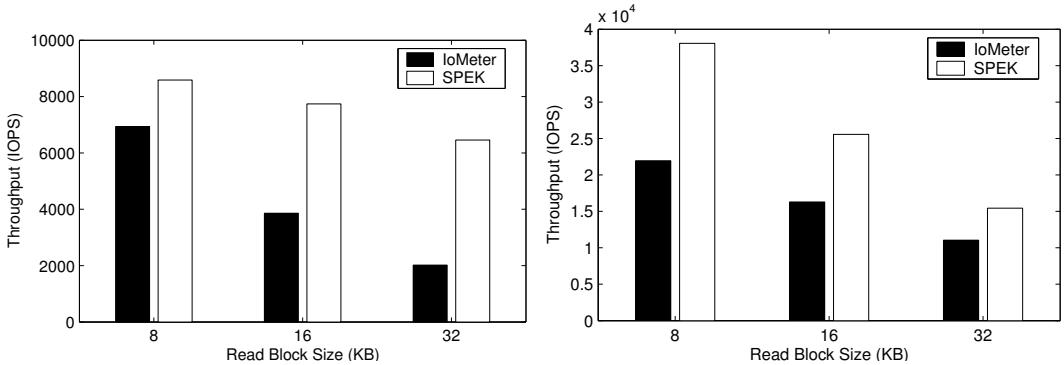


(a) Random Read Workload



(b) Sequential Write Workload

Figure 6. Measurement results on IBM SCSI disk with different workloads



(a) A storage system with 100 μs average response time (b) A storage system with 10 μs average response time

Figure 7. IoMeter and SPEK measurement results on simulated storage. When used to measure a storage system with 10 μs average response time (support 100k IO requests per second), SPEK generates more requests to saturate storage faster than IoMeter does because of lower overhead, which makes the measurement easier.

late them for our experimental purpose. Based on the Linux scsi_debug module, we have built a virtual SCSI disk device. When upper layer generates a read/write request to the virtual disk, the simulator simulates a disk delay time that is user configurable. Figure 7 shows the measurement results of the simulated high performance disk storages with disk access time being 100 microseconds and 10 microseconds, respectively. As expected, file system overheads result in much less throughput measured using IoMeter than that measured using SPEK.

In addition to affecting the accuracy of performance measurements, file system overheads can also lower the efficiency of the measurements. Such low efficiency may end up with longer time to perform a performance measurement or require more resources to carry out a same experiment. For example, if we were to measure the performance of an entry level RAID system as a networked storage as shown in Figure 7a, two SPEK Workers would be sufficient to saturate such a storage using SPEK while five workers would be necessary to do the same using IoMeter. Readers may wonder how much file system overhead is there using IoMeter as opposed to using SPEK. To give a quantitative view of such file system overheads, we measured number of context switches as well as number of system calls generated by the two benchmark tools. Table 3 lists the average number of context switches per I/O request with IoMeter and SPEK, respectively. As shown in this table, the average numbers of context switches per I/O generated by IoMeter and SPEK are 4.85 and 2.01, respectively. In terms of number of system calls per I/O request, we found that an IoMeter worker generates about 14 system calls on average for each I/O request while SPEK does not generate any system call because it is a kernel module. We used the HBench-OS [14] to measure context switches and system calls' overheads on our test clients. Each context switch cost ranges from 1.14 ms to 7.41 ms (average 4.27 ms) with different processes number and context related data size. The costs of six typical system calls, including getpid, getrusage, gettimeofday, sbrk, sigaction, and write, are 0.352ms, 0.579ms, 0.517ms, 0.036ms, 0.696ms, and 0.465ms respectively, with an average cost of 0.440ms. So for each IO request, IoMeter has approximately 19ms more overheads than SPEK that is comparable with the average response time of a high end RAID system, for example 10ms of a RamSan-210. This overhead hampers IoMeter when measuring a high end storage which is verified by Figure 7. So we believe that, SPEK is especially efficient when measuring high performance block storage systems. And this context switch and system call overhead reduction also explains why SPEK is superior than some benchmark tools that utilize the OS-provided raw access interface and run at user space.

	Sequential		Random	
	Read	Write	Read	Write
IoMeter	4.09	3.91	6.18	5.24
SPEK	1.98	2.01	2.03	2.02

Table 3. Average context switch numbers per I/O request of IoMeter and SPEK

	Client 1	Client 2	Client 3	Target
Test 1	18.012	N/A	N/A	18.012
Test 2	15.488	13.519	N/A	29.007
Test 3	9.035	7.645	6.534	23.214

Table 4. Throughput (MB/s) measurement of iSCSI SAN using SPEK with sequential read workload and block size being 32KB

To demonstrate that SPEK is capable of measuring networked storage, we have also carried out experiments on networked storages. Since we do not have SAN hardware facilities in our lab, we setup an iSCSI SAN [15] environment for measurement purpose. We use DISKIO mode in the iSCSI target allowing it to read/write Seagate SCSI disks. The iSCSI target exports several SCSI devices for test clients. We run different numbers of test clients using sequential read workload with block size being 32KB and results are shown in Table 4. We found that iSCSI target is saturated at 29.007MB/s using two test clients. Since it is a software iSCSI implementation, the TCP/IP and iSCSI protocol overheads [16] are the main reason why the target saturates rapidly. The CPU utilization of iSCSI target is consistently larger than 90% when using two test clients and approaches 100% when using three test clients. Majority of the time is consumed on iSCSI sending thread since for these read operations the target needs to send data out to clients.

4. Related Work

There are many I/O benchmark tools available to measure I/O performance. Typical benchmark tools fall into three categories as shown in Table 5.

Majority of I/O benchmark tools available today fall into file system benchmark category. Most of them create one or several files and perform read, write, append, and other operations on these files. Bonnie++ also has tests for file create, stat, and unlink operations. IOStone [17] only performs operations on a 1MB size file, which makes it impossible to get realistic results on modern storage systems because of large amount of file system cache. IOBench is obsolete and rarely used today. IoZone and IoMeter are the most popular among these benchmarks since they support many platforms and different file systems including network file system. IoZone is also famous for standalone file system benchmark allowing extensive file operations including read, write, re-read, re-write, read backwards, read strided, fread, fwrite, random read, pread ,mmap, aio_read, and aio_write. It reports throughput and response time results. IoMeter is originally from Intel and now a sourceforge project. It is being widely used and its workloads are highly parameterizable and configurable. While it is claimed to be a raw device test tool, IoMeter is still influenced by buffer cache under Linux as evidenced in the previous section. LADDIS [18] and SPEC SFS [19] only operate on NFS while NetBench [20] operates only on CIFS. PostMark [7] is also a widely used [6, 21] file system benchmark tool from Network Appliance. It measures performance in terms of transaction rates in an ephemeral small-file environment by creating a large pool of continually changing files. Pablo I/O Benchmark can be used to test the MPI I/O performance as well as application I/O but still at file system level. Its I/O Trace Library is very useful for analyzing application I/O behaviors while not aimed at block I/O measurement. NHT-1 I/O benchmark [22] measures application I/O, disk I/O, and network I/O, but its disk I/O measurement is still at file system layer.

Many of above mentioned benchmark tools perform well when used to measure file systems, which is the main purpose that they focus on. To measure disk I/O and block level networked storage system devices, all of them have the common problems as we have mentioned before. IoMeter suffers less because it operates on block device layer,

Category	Benchmark Tools
File System Benchmark	Bonnie, Bonnie++, IoMeter, IoZone, LADDIS, NetBench, PostMark, SPEC SFS, IOGen, IOStone, IOBench, LMBench, Pablo I/O Benchmark, NHT-1 I/O Benchmarks, NTIOgen, VxBench
Standalone Disk I/O Benchmark	CORETest, Disktest, HD Tach, QBench, RawIO, SCSITool
Block level Networked Storage Benchmark	SPC-1, SPEK

Table 5. I/O benchmark tools

which bypasses file system cache but it still suffers from buffer cache.

There are also a few benchmark tools for measuring block level or raw performance of a storage device. CORETest is a DOS disk benchmark tool from CORE International and it is rarely used now. Disktest can be used to perform disk I/O performance test but its main purpose is to detect defects. Qbench is a DOS hard disk benchmark from Quantum Corporation that measures data access time and data transfer rate. SCSITOOL is a diagnostics and benchmarking tool for SCSI storage devices. Pablo Physical I/O Characterization Tool [23], although not a benchmark tool, can be used to get useful trace information about disk I/O by using instrumented disk device driver. There are also some micro-benchmarks used in research works [24, 25]. Most of them are built to test some simple and limited I/O workloads, such as sequential read/write or random I/O in fixed sizes and aimed at standalone storage systems.

None of the existing benchmarks discussed above is able to measure performance of networked storage at block level. One specification, SPC-1, is aimed at measuring block level networked storages [26]. SPC-1 is a standard specification being considered by Storage Performance Council. It is not yet readily available to public for performance evaluation purposes although there are some incomplete performance data reported on the web. There are also limitations in using SPC-1 such as limitation on I/O streams and characteristics of each I/O stream [26]. To the best of our knowledge, our SPEK is the first benchmark tool for measuring block level performance of both DAS and networked storages with high flexibility, accuracy, and efficiency.

Besides performance benchmarking, some research work [27, 28, 29] concentrate on SCSI disk drive modeling. Their interests are accurately capture detail drive specification to support better scheduling or build disk drive simulator. Some performance characteristics such as path transfer rate, controller latency, and rotation latency can be extracted from their modelings. Researchers [30] also propose a method to improve file system performance by exploiting storage characteristics.

5. Conclusions

In this paper, we have proposed a new benchmark tool for block level performance evaluation of storage systems named SPEK, Storage Performance Evaluation Kernel module. SPEK can be used to accurately measure both DAS and networked storage with minimum influence of file system and low-level buffer caches. Performance results measured using our SPEK represent realistic and true performance of data storages. Users can easily configure SPEK to allow variety of workload characteristics to be tested and

to collect performance metrics of interest among a number of produced parameters. Because it runs as a kernel module, system overheads such as system calls and context switching are minimized making SPEK a highly efficient benchmarking tool. A Linux kernel module of SPEK has been implemented to demonstrate its functionality and effectiveness.

We plan to make the source code available to the public as soon as possible. Once finished, the preliminary SPEK code can be downloaded from our web site (http://www.ele.uri.edu/research/hpcl/).

Acknowledgments

This research is sponsored in part by National Science Foundation under grants MIP-9714370 and CCR-0073377. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. The third author's research is partially supported by the Manufacturing Center at Tennessee Tech University.

References

[1] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, "RAID : High-performance, reliable secondary storage," *ACM Computing Surveys*, vol. 26, no. 2, pp. 145–188, June 1994.

[2] G. Gibson and R. Meter, "Network attached storage architecture," in *Communications of the ACM*, vol. 43, Nov. 2000, pp. 37–45.

[3] E. L. Miller, W. E. Freeman, D. D. E. Long, and B. C. Reed, "Strong security for network-attached storage," in *Proc. of the Conference on Fast and Storage Technologies*, Monterey, CA, Jan. 2002, pp. 1–14.

[4] G. T. R. Khattar, M. Murphy and K. Nystrom, "Introduction to storage area network," Redbooks Publications (IBM), Tech. Rep. SG24-5470-00, Sept. 1999.

[5] J. Satran, K. Meth, C. Sapuntzakis, M. Chadalapaka, and E. Zeidner. (2002) iSCSI draft standard. http://www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-18.txt.

[6] K. Magoutis, S. Addetia, A. Fedorova, M. Seltzer, J. Chase, A. Gallatin, R. Kisley, R. Wickremesinghe, and E. Gabber, "Structure and performance of the direct access file system(DAFS)," in *Proceedings of USENIX 2002 Annual Technical Conference*, Monterey, CA, June 2002, pp. 1–14.

[7] J. Katcher, "PostMark: A new file system benchmark," Network Appliance, Tech. Rep. 3022, 1997.

[8] D. Capps and W. D. Norcott. Iozone filesystem benchmark. [Online]. Available: http://www.iozone.org/

[9] R. Coker. Bonnie++ benchmark tool. [Online]. Available: http://www.coker.com.au/bonnie++/

[10] Intel. Iometer, performance analysis tool. [Online]. Available: http://www.intel.com/design/servers/devtools/iometer/

[11] K. A. Smith and M. I. Seltzer, "File system aging - increasing the relevance of file system benchmarks," in *Proceedings of the 1997 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, June 1997, pp. 203–213.

[12] *SCSI Block Commands*, NCITS Working Draft Proposed Standard, Rev. 8c, 1997. [Online]. Available: http://www.t10.org/scsi-3.htm

[13] Dell/EMC. CX200 RAID storage system. http://www.dell.com/.

[14] A. Brown and M. Seltzer, "Operating system benchmarking in the wake of lmbench: A case study of the performance of netbsd on the intel x86 architecture," in *Proceedings of the 1997 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Seattle, USA, June 1997, pp. 214–224.

[15] UNH. iSCSI reference implementation. [Online]. Available: http://www.iol.unh.edu/consortiums/iscsi/

[16] X. He, Q. Yang, and M. Zhang, "Introducing SCSI-To-IP cache for storage area networks," in *Proceedings of the 2002 International Conference on Parallel Processing*, Vancouver, Canada, Aug. 2002, pp. 203–210.

[17] A. Park and J. C. Becker, "IOStone: a synthetic file system benchmark," *Computer Architecture News*, vol. 18, no. 2, pp. 45–52, June 1990.

[18] M. Wittle and B. E. Keith, "LADDIS: The next generation in NFS file server benchmarking," in *In USENIX Association Conference Proceedings '93*, April 1993.

[19] SPEC. SPEC SFS benchmark. http://www.spec.org/osg/sfs97/.

[20] VeriTest. Netbench file system benchamrk. http://www.etestinglabs.com/benchmarks/netbench/netbench.asp.

[21] J. L. Griffin, J. Schindler, S. W. Schlosser, J. S. Bucy, and G. R. Ganger, "Timing-accurate storage emulation," in *Proceedings of the Conference on File and Storage Technologies (FAST)*, Monterey, CA, Jan. 2002, pp. 75–88.

[22] R. Carter, B. Ciotti, S. Fineberg, and B. Nitzberg, "NHT-1 I/O benchmarks," NAS Systems Division, NASA Ames, Tech. Rep. RND-92-016, Nov 1992.

[23] H. Simitci and D. A. Reed, "A comparison of logical and physical parallel I/O patterns," *The International Journal of Supercomputer Applications and High Performance Computing*, vol. 12, no. 3, pp. 364–380, Fall 1998.

[24] Y. Zhu and Y. Hu, "Can large disk built-in caches really improve system performance?" University of Cincinnati, Tech. Rep. 259, 2002.

[25] E. Zadok and J. Nieh, "FiST: A language for stackable file systems," in *Proceedings of the 2000 USENIX Annual Technical Conference, San Diego, CA*, June 2000, pp. 55–70.

[26] SPC. SPC benchmark 1(SPC-1) specification. http://www.storageperformance.org/Specifications/SPC-1_v150.pdf.

[27] C. Ruemmler and J. Wilkes, "An introduction to disk drive modeling," *IEEE Computer*, vol. 27, no. 3, pp. 17–29, Mar. 1994.

[28] B. L. Worthington, G. R. Ganger, Y. N. Patt, and J. Wilkes, "On-line extraction of SCSI disk drive paramaters," in *Proceedings of Sigmetrics 95 / Performance 95*, Ottawa, Ontario, Canada, May 1995, pp. 146–156, published as Performance Evaluation Review Special Issue, Volume 23,No.1.

[29] "Automated disk drive characterization," CMU SCS, Tech. Rep. CMU-CS-99-176, Dec. 1999.

[30] J. Schindler, J. Griffin, C. Lumb, and G. Ganger, "Track-aligned extents: matching access patterns to disk drive characteristics," in *Proceedings of the Conference on File and Storage Technologies*, Monterey, CA, Jan. 2002, pp. 259–274.