An Extensible I/O Performance Analysis Framework for Distributed Environments

Benjamin Eckart[1], Xubin He[1], Hong Ong[2], and Stephen L. Scott[2]

[1] Tennessee Technological University, Cookeville, TN
[2] Oak Ridge National Laboratory, Oak Ridge, TN

Abstract. As distributed systems increase in both popularity and scale, it becomes increasingly important to understand as well as to systematically identify performance anomalies and potential opportunities for optimization. However, large scale distributed systems are often complex and non-deterministic due to hardware and software heterogeneity and configurable runtime options that may boost or diminish performance. It is therefore important to be able to disseminate and present the information gleaned from a local system under a common evaluation methodology so that such efforts can be valuable in one environment and provide general guidelines for other environments. Evaluation methodologies can conveniently be encapsulated inside of a common analysis framework that serves as an outer layer upon which appropriate experimental design and relevant workloads (benchmarking and profiling applications) can be supported.

In this paper we present ExPerT, an *Ex*tensible *Per*formance *T*oolkit. ExPerT defines a flexible framework from which a set of benchmarking, tracing, and profiling applications can be correlated together in a unified interface. The framework consists primarily of two parts: an extensible module for profiling and benchmarking support, and a unified data discovery tool for information gathering and parsing. We include a case study of disk I/O performance in virtualized distributed environments which demonstrates the flexibility of our framework for selecting benchmark suite, creating experimental design, and performing analysis.

Keywords: I/O profiling, Disk I/O, Distributed Systems, Virtualization.

1 Introduction

Modern computing paradigms, of which distributed computing represents perhaps the most large-scale, tend to be extremely complex due to the the vast amount of subsystem interactions. It is no surprise then that it is often very difficult to pinpoint how a change in system configuration can affect the overall performance of the system. Even the concept of performance itself can be subject to scrutiny when considering the complexities of subsystem interactions and the many ways in which performance metrics can be defined. Such varying definitions coupled with the combinatorially large array of potential system configurations

H. Sips, D. Epema, and H.-X. Lin (Eds.): Euro-Par 2009, LNCS 5704, pp. 57–68, 2009.

create a very hard problem for users wanting the very best performance out of a system.

A potential solution then is to provide a framework flexible enough to accommodate changing technologies and to be able to integrate advancements in I/O performance analysis in a complementary way. Many complex programs exist to benchmark and profile systems. There are statistical profilers, such as OProfile [1], sysprof [2], qprof [3], source level debugging tools, such as gprof [4], system-level statistics aggregators, such as vmstat [5] and the sysstat suite [6], and various macro, micro and nano-benchmarking tools, such as Iozone [7], lmbench [8], bonnie++ [9], iperf [10], and dbench [11]. All of these tools measure different computing elements. It would be convenient to have a common framework through which results from any test could be aggregated in a consistent and meaningful way. Furthermore, such a framework would be able to avoid "re-inventing the wheel" since it should be able to accommodate new developments in a modular fashion. *In short, our motivation is toward a software framework that would provide the flexibility to support relevant workloads, appropriate experiments, and standard analysis.* As of this writing, we are unaware of any framework currently exists for these sort of tests, but such a tool, if properly designed, could greatly help our understanding of such systems.

Our contributions detailed in this paper include the following: We design a flexible, extensible, and unified framework for I/O characterization in distributed environments. ExPerT facilitates system-wide benchmarking and profiling and provides tools for analysis and presentation, such as automatic graph generation, detailed performance statistics, and other methods of data discovery. The design itself is modular and extensible, being able to incorporate current state of the art benchmarking and profiling tools. ExPerT is designed to be usable in large distributed or high performance environments, with jobs transparently able to be executed in parallel and on different machines using lightweight TCP clients dispersed to each node.

The rest of this paper continues as follows: In Section 2 we discuss the design goals, implementation, and functionality of the framework. In Section 3, we use ExPerT to do preliminary analysis of disk I/O on both the KVM and Xen virtualization platforms. Section 4 introduces related work in the areas of benchmarking and analysis. Lastly, in Section 5, we give our conclusions.

2 I/O Characterization Framework

With goals of modularity and extensibility, we constructed ExPerT to support multiple profilers and benchmarking applications.

Our framework consists primarily of two components: a testing and batch creation tool, (*batch*), and a data discovery tool, (*mine*). A high-level view of these components can be seen in Figure 1. The *batch* module serves as a wrapper around the *test* module. Single tests can then be represented simply as a singleton batch test. The *mine* module parses and interactively enables a user to produce graphs and statistics from the tests.

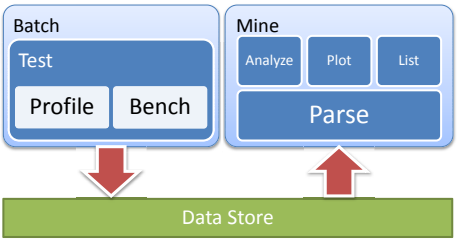

Fig. 1. Two main components comprise the framework: *batch* deals with the production of data. *mine* deals with aspects related to the consumption of the data: data parsing, trend discovery, and graphical plotting tools. Both share data via the lightweight database, *sqlite*.

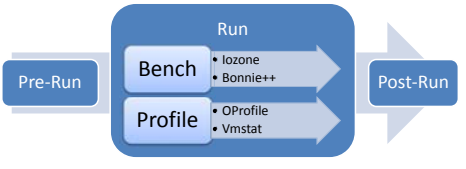

Fig. 2. The *batch* component: *batch* serves as a wrapper around *test*, which is itself comprised of three components, *prerun*, *run*, and *postrun*. *run* consists of an array of threads with a single master and n slaves. When the master stops, it signals the other threads to stop in unison. The threads comprising *run*, along with the *prerun* and *postrun* sections, can be a mixture of local and remote operations.

A graphical overview of the batch testing tool is shown in Figure 2. The testing tool defines three main functions that occur for each benchmarking or profiling application. First, a sequential list of commands can optionally be executed. This constitutes the *prerun* segment. The *prerun* module can contain any number of local or remote commands that prepare the system for the test. An example for disk I/O testing would be a *dd* command that constructs a file for benchmarking disk read throughput. The *run* segment starts an array of threads simultaneously, with each given local or remote commands for profiling and benchmarking. The threads follow a strict hierarchy: all threads begin at the same time, but all but one are designated to be "slave" threads. No threads halt execution until the single "master" thread halts. Since the master thread is designed to coordinate all the other threads synchronously, it suitably should be some terminal benchmark or application that is set to halt after a predetermined amount of time. The slave threads typically include profilers and tracers. Thus, we can "benchmark" benchmarks as a way of aggregating statistics from different tools. For example, we can run a single disk I/O test in *Iozone* and simultaneously watch the context-switching effects with *vmstat*. Finally, we conclude the test with another batch of sequentially executed commands. The implementation of this synchronization functionality utilizes a lightweight TCP client dispersed to each node.

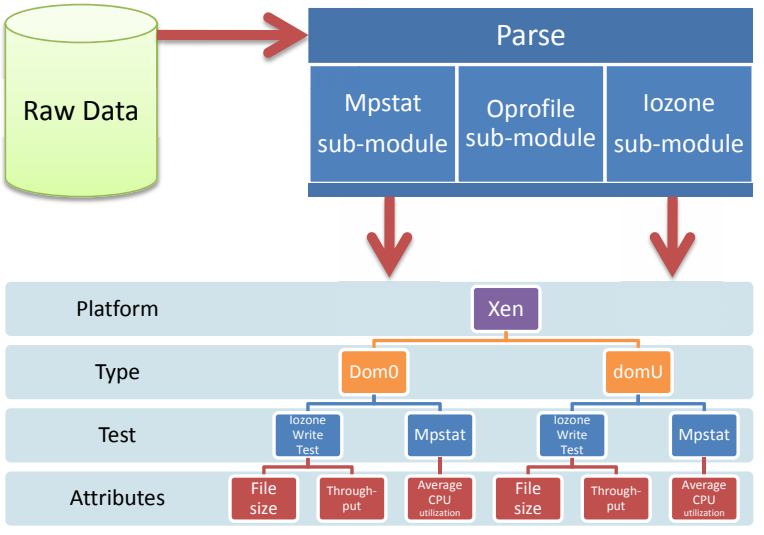

Fig. 3. The parsing sub-module of the *mine* component: *mine* deals with the parsing and presentation of the data gleaned by *batch*. The *parse* sub-module creates attributes dynamically from the data stored by sqlite. It then stores the parsed data back into temporary tables and passes it to one of several modules designed for aiding data discovery.

Mine aids in organizing, presenting, and analyzing the data collected by *batch*. A single test is represented as a datapoint containing various attributes pertinent to the application it involved. For example, a disk write test may record the file size, record size, operating system version, current disk scheduler, throughput, and so on. For the same test, the profiler employed may also record CPU and memory usage statistics. The only requirement for collecting the data is that the attributes, when parsed, have take the form of a few standard data types. An example of this parsing process is depicted in Figure 3.When aggregated into a single datapoint and combined with a multitude of other tests, we can begin to mine the data for trends and produce detailed statistics. Given a series of tests, for instance, we may choose to plot CPU usage and throughput versus varying record sizes. We can break this down even further by separating tests on the type of disk scheduler used. We can easily aggregate different batches in this way, since the batch itself is represented as just another attribute. All of these options have the same effect of reducing the dimensionality of the data to a point where results can be analyzed statistically or graphically plotted. We find this way of representing complex data to be very flexible. For example, if we switch from *vmstat* to *iostat* in our testing, we do not have to worry about creating new rules governing the new attributes that can be gleaned from this test. We can just record the attributes, and using the extensibility of the model, we can then choose the new attributes during the mining process. An example plotting wizard session is shown in Figure 4.

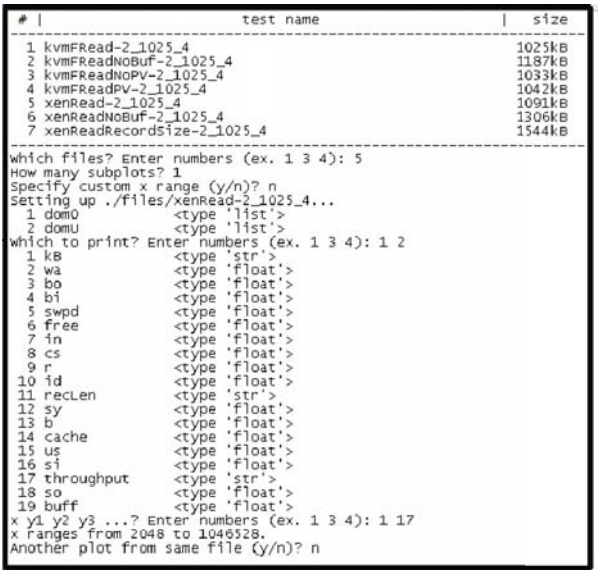

Fig. 4. An example wizard session for the mine tool. Tests are selected, parsed, and mined according to the specific traces the user wishes to see. At the end of the wizard, a graph is automatically generated. The graph generated by this session would look similar to the first subplot of Figure 6.

2.1 Implementation

We decided to use Python as our sole language, due to its object-oriented features and rising ubiquity as a high-level scripting language. In order to provide scalability and functionality on distributed systems, we disperse small TCP servers that coordinate jobs among the nodes. We use the *pylab* package, specifically *matplotlib*, for our plotting utilities. For our backend, we obtain lightweight database support with the python-supported *sqlite* standalone database system.

Several plug-in profiling modules have already been created for use with ExPerT, with several more forthcoming. The current modules are:

- Several tools from the sysstat suite [6], including mpstat and iostat
- Vmstat, a virtual memory monitoring tool [5]
- OProfile, a system-wide statistical profiler for Linux [1]

The only precondition for integration within the framework is that it follow the *prerun*, *run*, *postrun* format. Table 1 shows typical ways that these tools can be fit into this model. The actual configuration is specified by two xml files: one containing the node login details for the tested machines and one containing the *batch* module configuration. An example of latter file is shown in 5. The *profile* and *bench* modules are specified by assigning tags, which serves to specify the tool involved and the node on which it is to run during the test. Since there are no *prerun* or *postrun* requirements, these tags can be left out. The *node* tag

```
<batch type='kvm'>
  <test>
    <profile>
      <title>virtual test</title>
      <name>vmstat</name>
      <args>1</args>
      <node>oscarnode1.qcluster</node>
    </profile>
    <bench>
      <title>virtual test</title>
      <name>iozone</name>
      <node>virtnode1</node>
    </bench>
  </test>
  <args>
    <bench argsTemplate='-i 1 -s %dm -r 512k -+n -f /tmp/io.tmp'>
      <start>2</start>
      <stop>1024</stop>
      <step>2</step>
    </bench>
    <profile argsTemplate='1' />
  </args>
</batch>
```

Fig. 5. An example xml file for running a series of batch tests. Note that the *node* tag can be a remote machine.

Table 1. An example of different module configurations

Application	Thread Type	Prerun	Run	Postrun
Iozone	master	dd if=/dev/random of=io.tmp bs=1024 count=1024	iozone -i 1 -+n -s 1024k -r 512k -f io.tmp	(none)
mpstat	slave	(none)	mpstat 1	kill mpstat
vmstat	slave	(none)	vmstat 1	kill vmstat
OProfile	slave	opcontrol –init opcontrol –reset	opcontrol –start	opcontrol –shutdown opreport -l

includes the name or IP of the machine. Leaving off this tag will run the test on localhost. The batch tags use C's *printf* syntax and set up how the tests are to iterate.

3 Experiments in Disk IO with KVM and Xen

Though ExPerT can be used for any distributed profiling applications, we decided to use ExPerT in our initial experiments to probe into the performance characterization of an ever-growing popular distributed system: virtual systems. Since virtual systems are typically made to communicate through the network stack, ExPerT can be naturally used for these types of applications without any modification. We decided to look into virtual disk I/O performance because it remains a relatively unstudied area, in contrast to virtual CPU performance or virtual network I/O performance. It has been shown that when running CPU-intensive benchmarks, performance nearly parallels that of the native execution environment. Virtual I/O performance is a difficult problem due to the complexities of page tables, context switching, and layered execution paths. Furthermore,

disk I/O is a vital operation for operations such as live migration, checkpointing, and other fault tolerant or data redundant strategies.

3.1 Experiments with ExPerT

Using ExPerT, we conducted a series of virtual disk I/O benchmarking experiments. Our hardware is composed of a Dell OptiPlex 745, equipped with an Intel Core 2 Duo 6400 with VT enabled, 2 Gigabytes of RAM, and a Barracuda 7200.10 SATA hard drive. Our software stack consists of the the Kernel-based Virtual Machine (KVM) [12] with Ubuntu 8.04 and Xen 3.2 [13]. For all our experiments, we used the Iozone application as our workloads.

KVM Disk I/O Paravirtualization. Beginning with the Linux kernel version 2.6.25, the *virtio-blk* module for paravirtual block I/O was included as a standard module. Using KVM (KVM-72), we looked for any performance benefits under Ubuntu 8.04 with a 2.6.26 kernel using this *virtio-blk* module. We did an extensive series of tests with ExPerT and found little performance difference with Iozone. A sample performance graph is shown in Figure 6 for the Read test. *wa*, *sy*, and *us* refer to I/O wait, system, and user percent processor utilization, respectively. *cs* is the number of average context switches per second. We did

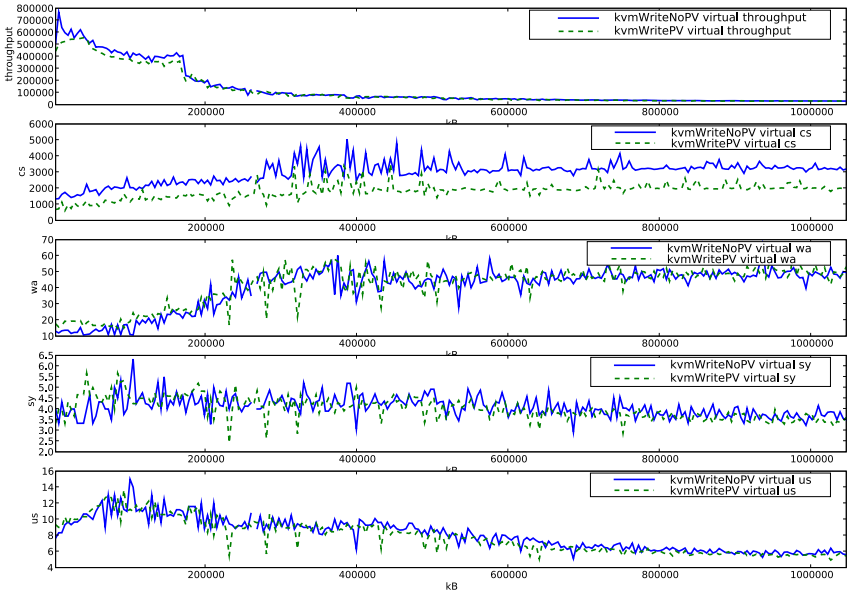

Fig. 6. KVM Paravirtualization: Tests with and without the paravirtual block I/O modules. Other than a small difference in context-switching performance, we could not see any real performance difference. We attribute this lack of improvement to the fact that the module is not very mature and has yet to be optimized.

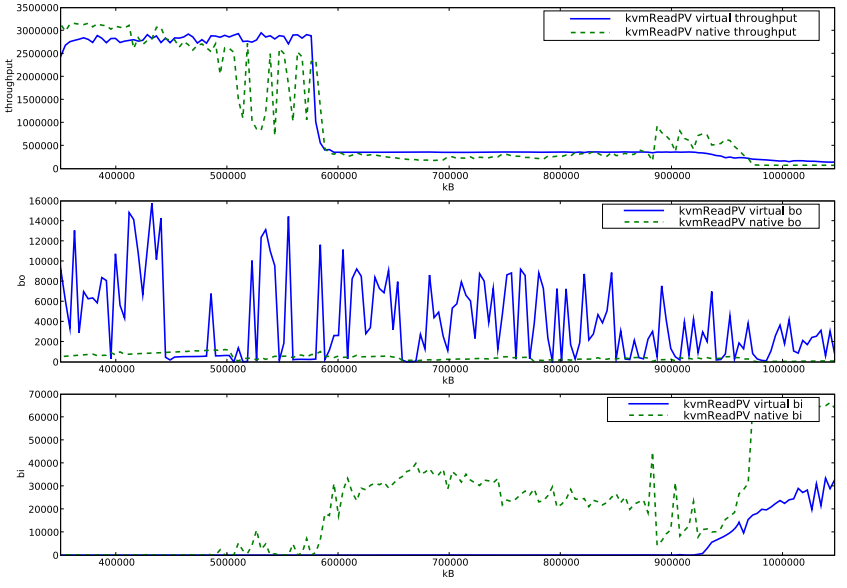

Fig. 7. KVM Guest Execution vs. Native Execution: Benchmarking and profiled memory statistics. The tags "bo" and "bi" represent the number of blocks per second (average) written out or read in from a block device (hard disk). The block size is 1024 bytes. The host is Ubuntu 8.04 Linux 2.6.25.9, and the guest is Ubuntu 8.04 Linux 2.6.26. Paravirtual I/O drivers were used.

find a small improvement in the number of average context switches per second reported by *vmbench*. We believe the module shows promise but it simply too immature and needs further optimization.

KVM Memory Management. Shown in Figure 7, we did a file read batch and looked at the average blocks per second read in from the disk and the average blocks per second written out to the disk. Since these statistics show the actual hard disk usage, we can immediately deduce the caching properties of the system. The results clearly show that for the native reads, the blocks read in from the disk start to increase about as the file size increases beyond the available free RAM. Another increase can be see when nearing 1 Gigabyte, which correlates with a drop in overall throughput. We can also see the caching mechanisms of KVM at work, only requiring significant reading from the disk at just over 900 Megabytes. Also, note that due to the copying between the guest and KVM, there is significant blocks out per second for all virtual tests.

KVM and OProfile. We also ran tests with OProfile integrated into the framework while running Iozone. We choose to look at the number of data TLB misses for KVM guests and for the native systems running the Iozone benchmark with

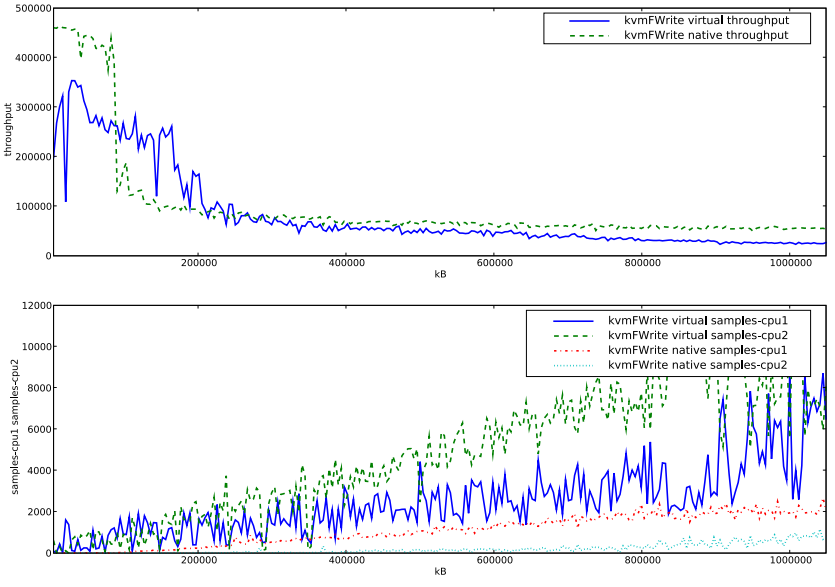

Fig. 8. KVM and OProfile: Detailed profiling statistics from OProfile while running Iozone. We chose to display the DTLB misses in this figure. The y-axis of the bottom subplot is in 1000's of misses.

the FWrite test. In the past, this has shown to be a source of significant overhead for virtual machines [14]. As expected, the number of DTLB misses grew substantially more when virtualized than the native testing. It is also curious that the DTLB misses were dominated by the first CPU. We found this to be the case across most OProfile tests.

KVM and Xen Performance. Finally, we detailed comparative performance tests of Xen and KVM. Figure 9 shows the results of three tests: FRead, FWrite, and Random, a random mix of reads and writes. Xen dom0 showed peculiarly good performance, even besting native performance. We expected good performance from Xen since it employs mature, paravirtualized I/O, but the performance is much better than we were expecting. The only type of disk operation where it wasn't the best was the 175-300 Megabyte range for file writing. Other than the dom0 outlier datapoints, we found Xen and KVM to be fairly comparable overall. The most drastic performance difference seem to come from differences in caching strategies, as evidenced by the varying plateau-like descents of the throughputs for the different file operations.

Though a more thorough analysis of this subject is still warranted, such analysis is beyond the scope of this paper, which it to present the toolkit and its capabilities, and provides the subject for future work.

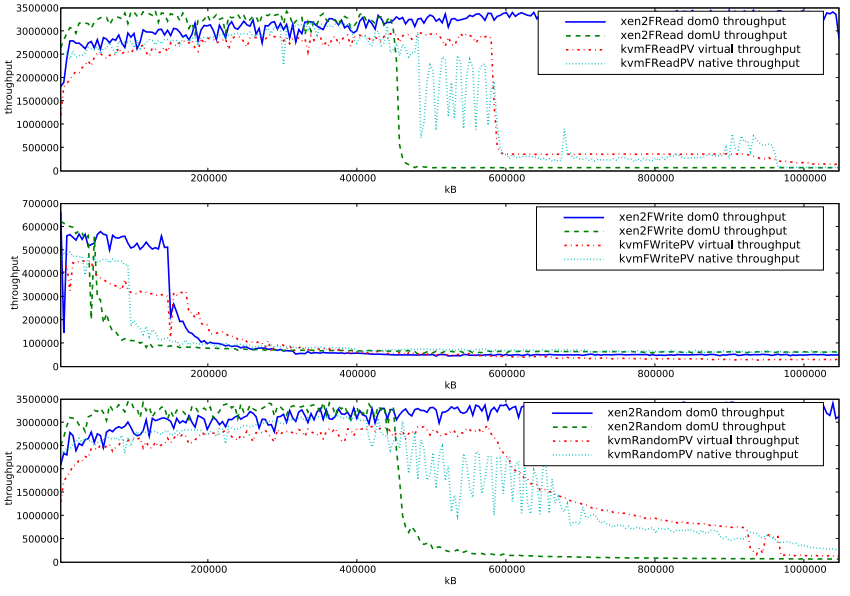

Fig. 9. KVM vs Xen: Read and Write operations for KVM, dom0, and domU

4 Related Work

Virtual disk I/O performance studies have been carried out before [15] as well as research dealing with virtualized storage [16], and though the focus of our paper is primarily to describe our framework, our case study deals closely with other notable attempts to characterize virtual disk I/O, including recent comparisons between KVM and Xen [17].

In the area of I/O benchmarking, there exist many different strategies and applications. For virtual systems, the SPEC virtualization committee has been working on a new standard benchmarking application [18]. Other virtual benchmarks include vmbench [19] and VMmark [20], a benchmarking application made by VMware. There is a wealth of research and toolkits on general I/O benchmarking techniques as well [21,22]. Finally, there exist anomaly characterization methods used in conjunction with I/O benchmarking for the purposes of pinpointing bugs and finding areas of improvement [23].

ExPerT is not a benchmark itself, but a characterization framework that integrates benchmarking, experimental design, and standard analysis. The most closely related projects to ExPerT include the Phoronix Test Suite, a general benchmarking framework [24], and Cbench, a cluster benchmarking framework [25]. The Phoronix Test Suite is limited in representation and functionality and relies heavily on the idea of community results distributed via the web. Some of the ideas behind Cbench are shared with ExPerT; Cbench places an emphasis on supporting an array of various benchmarking applications, and to a certain

extent, provides a common way to display results. In contrast, ExPerT centers around detailed workload characterization, standard analysis tools, and support for dynamic reconfiguration. Thus, the functionality of Cbench is encapsulated by the workload characterization portions of ExPerT, allowing Cbench to be used as the workloads component itself.

5 Conclusion

ExPerT is a broad, versatile tool for I/O characterization in distributed environments, facilitating benchmarking and profiling, and analysis and presentation. ExPerT consists primarily of two main modules: *batch* for automated batch testing and *mine* for automated data discovery, analysis, and presentation. Following the design principles of modularity, extensiblity, and consistency, we have built ExPerT as a tool that eases the task of analysis and characterization of I/O on distributed systems. We have designed ExPerT to be a natural fit for large distributed systems, transparently employing remote connections to distribute jobs across several machines and in parallel. In our sample comparison study of virtual disk I/O in typical virtual distributed systems (KVM and Xen), ExPert proves to be efficient to gather, sift, and analyze the large quantities of information that result from extensive testing. We believe ExPerT shows much promise as an I/O characterization and analysis framework for distributed environments. For more information, the ExPerT tool is available on the website http://iweb.tntech.edu/hexb/ExPerT.tgz.

Acknowledgment

This research was supported by the U.S. National Science Foundation under Grant No. CNS-0720617 and by the Center for Manufacturing Research of the Tennessee Technological University.

References

1. OProfile: A system-wide profiler for linux, `http://oprofile.sourceforge.net`
2. Sandmann, S.: Sysprof: a system-wide linux profiler, `http://www.daimi.au.dk/sandmann/sysprof`
3. Boehm, H.J.: The qprof project, `http://www.hpl.hp.com/research/linux/qprof/`
4. Graham, S.L., Kessler, P.B., Mckusick, M.K.: Gprof: A call graph execution profiler. In: SIGPLAN 1982: Proceedings of the 1982 SIGPLAN symposium on Compiler construction, pp. 120–126. ACM, New York (1982)
5. Vmstat: Vmstat man page, `http://www.linuxcommand.org/man_pages/vmstat8.html`
6. Godard, S.: Sysstat utilities homepage. `pagesperso-orange.fr/sebastien.godard/`
7. Iozone: Iozone file system benchmark, `www.iozone.org`

8. McVoy, L., Staelin, C.: lmbench: portable tools for performance analysis. In: ATEC 1996: Proceedings of the annual conference on USENIX Annual Technical Conference, Berkeley, CA, USA, pp. 23–23. USENIX Association (1996)
9. Coker, R.: The bonnie++ file-system benchmark, http://www.coker.com.au/bonnie++/
10. Tirumala, A., Qin, F., Ferguson, J.D.J., Gibbs, K.: Iperf-the tcp/udp bandwidth measurement tool, http://dast.nlanr.net/Projects/Iperf/
11. Dbench: The dbench benchmark, http://samba.org/ftp/tridge/dbench/
12. Kivity, A., Kamay, Y.D., Laor, U.L., Liguori, A.: kvm: the linux virtual machine monitor. In: OLS 2007: Proceedings of the 2007 Ottawa Linux Symposium, Ottawa, Ontario, Canada, pp. 225–230. USENIX Association (2007)
13. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: SOSP 2003: Proceedings of the nineteenth ACM symposium on Operating systems principles, pp. 164–177. ACM, New York (2003)
14. Menon, A., Santos, J.R., Turner, Y., Janakiraman, G.J., Zwaenepoel, W.: Diagnosing performance overheads in the xen virtual machine environment. In: VEE 2005: Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments, pp. 13–23. ACM, New York (2005)
15. Ahmad, I., Anderson, J., Holler, A., Kambo, R., Makhija, V.: An analysis of disk performance in vmware esx server virtual machines (October 2003)
16. Huang, L., Peng, G., cker Chiueh, T.: Multi-dimensional storage virtualization. In: SIGMETRICS 2004/Performance 2004: Proceedings of the joint international conference on Measurement and modeling of computer systems, pp. 14–24. ACM, New York (2004)
17. Deshane, T., Shepherd, Z., Matthews, J., Ben-Yehuda, M., Shah, A., Rao, B.: Quantitative comparison of xen and kvm. In: Xen Summit, Boston, MA, USA, June 2008, pp. 1–2. USENIX Association (June 2008)
18. Standard Performance Evaluation Corporation: Spec virtualization committee, http://www.spec.org/specvirtualization/
19. Moeller, K.T.: Virtual machine benchmarking (April 17, 2007)
20. Makhija, V., Herndon, B., Smith, P., Roderick, L., Zamost, E., Anderson, J.: Vmmark: A scalable benchmark for virtualized systems (2006)
21. Joukov, N., Wong, T., Zadok, E.: Accurate and efficient replaying of file system traces. In: FAST 2005: Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies, Berkeley, CA, USA, p.25. USENIX Association (2005)
22. Anderson, E., Kallahalla, M., Uysal, M., Swaminathan, R.: Buttress: A toolkit for flexible and high fidelity i/o benchmarking. In: FAST 2004: Proceedings of the 3rd USENIX Conference on File and Storage Technologies, Berkeley, CA, USA, pp. 45–58. USENIX Association (2004)
23. Shen, K., Zhong, M., Li, C.: I/o system performance debugging using model-driven anomaly characterization. In: FAST 2005: Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies, Berkeley, CA, USA, p. 23. USENIX Association (2005)
24. PTS: Phoronix test suite, www.phoronix-test-suite.com
25. Cbench: Scalable cluster benchmarking and testing, http://cbench.sourceforge.net/