Energy Inefficiency of Operating System Layers for Data-centric Infrastructures

Shoaib Akram, Manolis Marazakis, and Angelos Bilas[†]
Foundation for Research and Technology - Hellas (FORTH)
Institute of Computer Science (ICS)
100 N. Plastira Av., Vassilika Vouton, Heraklion, GR-70013, Greece
Email: {shbakram,maraz,bilas}@ics.forth.gr

1. ABSTRACT

In this work we examine the relative overhead of the operating system and using virtual machines on I/O intensive applications. We use real applications to calculate the cycles and energy per I/O using simple models, based on actual measurements. Our results indicate that the OS can cost up to 60% in terms of energy spent per I/O operation. Further, a single VM instance costs 150% in terms of performance and 180% in terms of energy consumption per I/O operation. For some applications, server consolidation using two VM instances can reduce the cost compared to one VM instance by up to 25%. Finally, we note that current system stacks do not scale with the number of cores and on average, compared with one core, the system component of execution time increases by 90x on a 1000-core processor.

2. INTRODUCTION

The amount of data produced in modern society is increasing rapidly. Various projections estimate that by 2020, the world will produce 35 Zeta Bytes of data [10]. To manage and process the huge datasets, the notion of data-centric applications has emerged [12]. Data-centres already consume a significant amount of energy and at the compound annual growth rate (CAGR) of about 14%, the total energy consumption of data-centres in United States will be 300 billion kilo Watt hours (kWh). Thus, there is increasing pressure to improve the efficiency of modern data-centric applications.

Many data-centric applications heavily use the operating system (OS) for access to network and storage. For many applications, the OS contribution to the total execution time is comparable to or higher than the user time i.e., the time an application spends executing code in the user space. In particular for data-centric applications, all work performed by the OS aims to provide name translation, recovery, buffer management, and device management; It is not related to the actual data itself, but rather to the process of moving data from persistent storage to application buffers.

[†]Also, with the Department of Computer Science, University of Crete, P.O. Box 2208, Heraklion, GR-71409, Greece.

In addition, today, virtual machines (VMs) are typically used to allow running multiple applications on the same server. Improving server utilization typically requires running multiple applications and for isolation purposes these are usually run within separate VMs. This introduces additional overheads for performing I/Os.

In this work we are interested in understanding the relative overhead of the OS and VMs over application processing for I/O intensive applications. We use applications and datasets that are typical of workloads deployed in today's data-centres. We run the workloads using a commodity OS and measure the overhead of OS alone. We use a simple model to translate the execution time breakdown to cycles and energy spent per I/O (cpio and eio). We then run the workloads using a popular Virtual Machine Monitor (VMM) and measure the additional overhead. We also report the reduction in energy cost per I/O when multiple VMs are used to run independent instances of the same workload.

We run each workload using configuration parameters and datasets that result in a large amount of I/O. We first run each workload on top of a commodity OS distribution with the latest Linux kernel and measure the breakdown of execution time. We report the overhead of the system component alone compared to the total execution time spent for processing purposes. We show that, although low compared to the user component, the overhead of the system component grows with the number of cores. Finally, we examine how overheads scale by looking at cpio and eio on an increasing number of cores.

We observe that compared to the energy consumed by the user component of a single instance of a workload:

- The servers that use disk-based storage subsystems, apart from slow response times, are also energy inefficient in processing I/Os. In particular with current system stacks, servers using disk-based storage subsystems spend up to 6x more energy per I/O operation. The gap is reduced from 6x to only 58% if the idle power consumption in servers is made 0% of peak power.
- On average, the system component of execution time costs 60% in terms of energy consumption per I/O operation.
- Virtualization costs up to 150% in terms of cycles spent

per I/O operation for a suite of data-centric application. Similarly, adding the VM layer costs, on average, a 180% overhead in terms of energy consumed per I/O operation.

- The OS layer, as it is today, does not scale with the number of cores. On average for eight data-centric workloads, there is a 12x, 24x, and 92x increase in system cycles per I/O operation compared the system cycles with one core.

The rest of the paper is organized as follows. Section 3 describes our methodology for collecting measurements and quantifying performance and energy efficiency. Section 4 describes the applications that we use for evaluation purposes. Section 5 describes the platform that we use for evaluation and the various component of execution time on two different server machines. The same section also provides baseline energy consumption per I/O operation on the two machines for the applications we use in this work. In Section 6, we discuss the energy inefficiency of OS and VM layers. We discuss related work in Section 8, and finally, we conclude this work in Section 9.

3. METHODOLOGY

Our main goal in this paper is to quantify the energy overheads introduced by the system layers. In particular, we are interested in the OS layer and the VM layer. Application-level metrics are thus not useful for analyzing individual components of today's complex software stacks. Therefore, we propose to use cycles per I/O (cpio) operation as a metric for quantifying system-level overheads. We calculate cpio by running the application and measuring the execution time breakdown consisting of user, system, idle and iowait time. Next, we briefly discuss what each component of the execution time means.

User time refers to the time an application spends executing code in the user space. When an application requests services by the OS, the time spent is classified as system time. The time an application spends waiting for a pending I/O operation to complete is called iowait time. Finally, when a processor has no work to perform, the time is counted as idle time.

We also note the number of 512 byte I/O operations that an application performs during the execution time. This is a typical size for a sector on many storage devices. Note that the actual size of I/O operation from the point of view of kernel is different and typically of the order of kilo bytes. We assume that the entire volume of data read/written during the execution time consists of many 512 byte chunks.

Next, we calculate cpio by dividing the user plus system cycles consumed during the execution time by the number of I/O operations. We do not run the workloads to completion but long enough to capture the representative behavior. In particular, we take measurements during which the application generate I/O throughput that is typical if the application is allowed to run for much longer periods.

An important benefit of cpio as a system-level metric is that it is easy to translate to energy per I/O operation. We use a simple model to calculate the energy spent per I/O operation. Our model is derived from the results reported in recent literature [9, 20, 18]. In particular, we measure the CPU utilization, and then use a peak power and idle power typical of our server machines to calculate the energy per I/O operation. We do not include the energy consumed by storage devices. We report results assuming no idle power and what is typical of servers deployed in today's data-centres [20].

We use the following equation to calculate energy per I/O (eio) in terms of watts.seconds or joules :

$$eio = \{P * (1 * cpio + IP * cpio^{i})\}/(N * F)$$

In the above equation, P is the peak power of the server machine (minus storage subsystem), cpio is the cycles per I/O, $cpio^{i}$ is the idle cycles per I/O plus iowait cycles per I/O, IP is the fraction of peak power when the machine is in idle state, N is the number of cores, and F is the frequency of each core. We show results for energy consumption required to perform (and process) a million I/O operations (emio).

A software stack can appear to be cpio- or eio-efficient by doing a large number of small I/O operations. For instance, applications that perform a large number of metadata operations throughout execution could appear to have a low cpio, although metadata I/Os are typically "induced" I/Os and not strictly necessary for processing a specific dataset. Thus, when looking at cpio and eio, their mere absolute value is not appropriate to judge the "quality" of a stack or platform.

In our work we first compare the emio for two server machines with different storage subsystems. We then calculate the emio without the OS layer. Next, we run the workloads inside a VM and show the increase in emio. We then use server consolidation using two VMs and show how much the energy consumption per I/O of running is amortized. In this work, we show emio for up to two VMs. Finally, we use the system component of cpio to evaluate the scalability of current OS stacks. We measure the system cycles per I/O for 1, 4, 8 , and 16 cores and project it using a linear model to 1000s of cores.

4. APPLICATIONS

In this section, we discuss the configuration parameters, mode of operation, and the datasets we use for the applications we use for evaluation. Table 1 summarizes the workloads using the applications. Note that in some cases, we use multiple instances of the same application to form a workload that better utilizes the server and storage resources. Next, we briefly discuss each application:

Checkpointing is done in high performance computing (HPC) applications to recover from possible failures. We use IOR [16] that simulates various checkpointing patterns that appear in the HPC domain. Due to the MPI layer, IOR has a moderate user time and in-flight I/O issued by several concurrent MPI processes induce significant iowait time due to contention for I/O resources.

File indexing is mainly done as a back-end job in data-centres and web hosting facilities. We use Psearchy [15, 5] as a file indexing application. We run Psearchy using multiple processes where aach process picks files from a shared queue of file names. A hash table is maintained by each process for storing BDB indices in memory. The hash tables are flushed to the storage devices once they reach a particular size. We modified original Psearchy to use block-oriented reads instead of character-oriented reads to improve I/O throughput. There is an increase in system time of Psearchy for

Table 1: Workloads for Evaluation.

Application	Description	Parameters	Dataset Size (GBytes)
IOR	Application Checkpointing	Processes=128; File Size=2GB; I/O Mode=MPI_IO; Offseting within File=Sequential	128
Psearchy	File Indexing	Directory Hierarchy=Flat Document Size=10MB Processes=32; Hash Table Size=128MB	100
Dedup	File Compression	Dedup: File Size=1GB; Instances=10; Threads per Stage=32	10
Metis	Mapreduce Library for Single Multi-core Machines	Application:Word Count; Instances=5 File Size=1GB; Map Threads=8; Reduce Threads=8	20
BDB	Key-value Store (Java-based)	Threads=128; Fields per Record=10; Size of Field=1KB;	30
HBase	NoSQL Store	Threads=128; Fields per Record=10; Size of Field=1KB	30
TPC-E	OLTP Workload (Stock Broker)	Active Customers=200000; Days of Trade=7; Terminals=128 innodb_thread_concurrency=8; innodb_file_io_threads=4	155
BLAST	Sequence Similarity Search	Instances=16; Threads per Instance=16; Task=*blastn* Queries per Instance=16; Databases=Pre-formatted Nucleotide-Databases from NCBI (refseq_genomic, env_nt, nt) Alignments=128; Target Sequences=5000	20

indexing large files. Conversely when indexing small files, there is an increase in user time. A complex directory structure results in an increase in system time.

Deduplication is a technique for compression used in storage farms and data-centres. We use the Dedup kernel from the PARSEC benchmark suite [4]. The Dedup kernel has five pipeline stages with the first and the last stage for performing I/O and the middle three stages can each use a pool of threads. We observed that the peak memory utilization of Dedup is very high for large files because intermediate queues between pipeline stages grow very large. In order to drain the queues quickly, a large number of threads is required in the intermediate pipeline stages. The memory utilization is not an issue for small files because there is less intermediate state to be maintained. Dedup is entirely dominated by user time.

Mapreduce is a programming model being increasingly used for developing data-centric applications. We use a Mapreduce programming library for single nodes called Metis that is part of the Mosbench benchmark suite [5]. Metis maps the input file in memory and assigns a portion of the file to each of the *map* threads. We use Metis to count words in a file. A single instance of Metis does not generate a lot of I/O and is primarily dominated by user time. We run multiple instance of Metis and assign a different file to each instance.

NoSQL data stores are becoming popular for serving data in a scalable manner. HBase is a non-relational data serving system that is part of the Hadoop framework. We use the YCSB framework [6] from Yahoo to test HBase. We first build a database using the YCSB load generator (using a workload that performs only insert operations). We then run a workload that performs 70% read operations and 30% upgrade operations. We reserve 3GB of physical memory for the Java virtual machine (JVM). HBase has high idle time while there is an equal amount of user and system time.

Key-value data stores remain a famous choice for serving data in the data-centres. BDB is a library that provides support for building data stores based on key-value pairs. The evaluation methodology for BDB is similar to that we use for HBase. Since BDB is an embedded data store, the YCSB clients and the BDB code shares the same process address space. Therefore, we reserve 6GB of physical memory for the JVM. We configure YCSB to use 3GB for the YCSB clients and 3GB for BDB. BDB is dominated by user time but there is considerable system time.

Online transaction processing (OLTP) is an important class of workloads for data-centres. We use TPC-E for evaluating an OLTP workload. TPC-E models transactions that take place in a stock brokerage firm. We run TPC-E using MySQL database system and specify runtime parameters that results in high concurrency. We observe that using MySQL database server results in high idle time for both TPC-C and TPC-E either due to non-scalability to multiple cores or idling due to synchronization among a large number of threads.

Comparative genomics leverages the tremendous amount of genomic data made possible by advances in sequencing technology. We use BLAST [2] for Nucleotide-Nucleotide sequence similarity search. We run multiple instances of BLAST each executing a different set of queries on a separate database. We use random query sequences of 5 KB which is a common case in proteome/genome homology searches. BLAST is I/O intensive and the execution time is primarily dominated by user time.

5. EVALUATION PLATFORMS AND IMPACT OF STORAGE SUBSYTEMS

We characterize the behavior of the workloads in Table 1 using a disk-based and an ssd-based server machine. Our main purpose for using two different servers is to observe how the execution breakdown of workloads is effected by the underlying storage hardware. The important features of the two machines we use for evaluation are listed in Table 2.

For experiments with virtualization, we use the virtualization infrastructure for the Linux kernel (KVM). We note that KVM requires that at least 1 core and up to 2 cores are reserved for the KVM for workloads to run properly. We perform virtualization experiments using only 8 (logical) processors and 8 GB of DRAM. Thus, we leave 8 logical processors for KVM and 4 GB of DRAM for the host kernel. When we run two VMs, we allocate 4 logical processors and 4 GB of DRAM for each VM instance. Similarly, we partition our storage subsystem on SSDS into two RAID 0 devices each consisting of 12 SSDs and assign one for each of the two VM instances.

First, we note that on average, the system time on DISKS

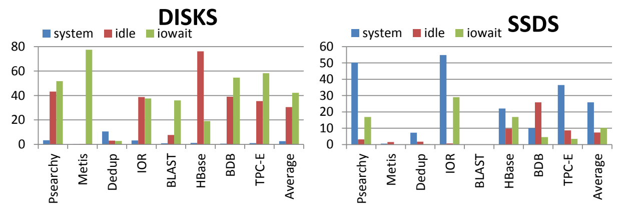

Figure 1: Percentage of execution time (Y-axis) spent as system, idle and iowait time on SSDS and DISKS.

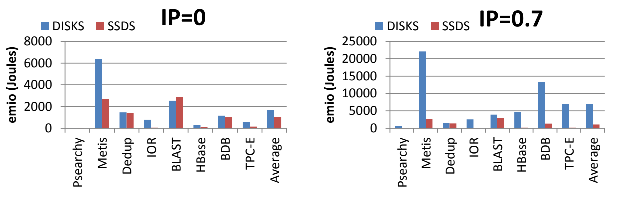

Figure 2: Energy Consumption per Million I/Os (emio) on DISKS and SSDS.

is only 2% of total execution time compared with 26% on SSDS. However, the more subtle result is that the cycles spent per I/O (cpio) and in particular the system cycles per I/O is almost the same for the two machines. Thus the efficiency of the system stack i.e., the OS and other layers below the user application is still relevant for both machines. However, to understand the overheads introduced by the addition of VM layer, we report results for the VM layer using SSDS. Also, we perform the scalability experiments and report results for SSDS. In essence, scalability trends are easier to identify and project when I/O is not the bottleneck.

The power estimation of mid-range server machines today is roughly 400 Watts which is the value we use for our energy calculations. Note that since cpio is largely independent of a particular server machine, the specific power we use to calculate eio (and emio) is not important for comparing the various system layers.

Next, Figure 1 shows the breakdown of execution time in terms of user, system, idle, and iowait time. Note that, as expected, the machine with disk-based storage subsystem has high iowait time. On average, the iowait time on DISKS is 42% of total execution time compared with 10% on SSDS. However, we note that the idle time is also high for the disk-based machine. On average, the percentage of idle time is 30% on DISKS compared to only 7% on SSDS. This is because, typically in application today, there are threads whose main task is to perform I/O and distribute work or provide data to a large number of threads that perform the main application functionality. If the I/O subsystem is slow, the threads performing I/O have less work to distribute to other threads and thus the entire system is inefficient or there is more idle time.

This observation is of particular interest to data-centric infrastructures as in today's server machines, the idle and iowait periods consume up to 70%of the peak power. This implies that disk-based storage subsystems not only have slower response time but are also inefficient in processing I/Os in terms of energy. Figure 2 shows the emio for both machines. We show the emio for both machines assuming idle power (IP) of 0% and 70% of peak power. First, we note that on average with IP=0, emio for DISKS is 58% greater to that of SSDS. However, with IP=0.7, emio for DISKS is 6x to that of SSDS. We conclude that for today's servers where IP is more close to 0.7, it is inefficient in terms of energy to operate application using storage subsystems with low throughput. However, given idle power consumption is reduced close to 0% of peak power, the gap in terms of energy inefficiency will be reduced greatly.

For DISKS, idle and iowait cycles cost significantly in terms of energy consumption. On average, an IP of 0.7 compared to 0 results in a 4x increase in emio for DISKS.

Table 2: Summary of Machine Parameters.

DISKS	SSDS
2 Intel Xeon E5620 (Quad-core)	2 Intel Xeon E5405 (Quad-core)
No Hyper-Threading	2 Hardware Thread per Core
8 GB RAM; 1 Storage Controller	12 GB RAM; 4 Storage Controllers
XFS on Hardware RAID 0 (8 Disks)	XFS on Software RAID 0 (24 SSDs)
Storage Throughput=1 GB/s	Storage Throughput=6 GB/s
CentOS distribution; 2.6.18 kernel	CentOS distribution; 2.6.32 kernel

We make an effort in this work to run all workloads with maximum concurrency to maximize the server utilization. In certain cases, where interference and contention is not an issue, we run multiple instances of the same application. Therefore, idle cycles for the workloads evaluated in this work is an application behavior. Apart from excessive idle cycles on DISKS, this behavior has implications also on the SSDS machine. For instance, both HBase and BDB does not fully utilize the server resources on both SSDS and DISKS and thus there is a noticeable difference between the emio with IP of 0 and IP of 0.7.

6. QUANTIFYING OVERHEADS OF VARIOUS LAYERS

In this section, we compare the emio for the user component of application alone, the user plus the system component together, the same application running within a VM, and two instances of the same application running in two separate VMs. All results in this section are shown for an IP of 0.7.

First, Figure 3 compares the emio with and without the system overhead for workloads with significant system time (10% or more on SSDS). We note that for DISKS, there is no difference in emio with or without the system overhead. This is because the execution time is dominated by either idle and/or iowait time. However, on SSDS, the introduction of the system layer results in a 60% increase in emio (on average for five applications). The maximum overhead is for Psearchy (250%). Note that Psearchy is a simple application but ends up consuming large amount of energy mainly for accessing the files to build indices. Although it is not realistic to assume that the system-level overheads will be completely eliminated but this work shows the range by which modern stacks could be improved for energy efficiency.

Next, we discuss and compare the results with virtualization. Figure 4 shows the overhead due to virtualization both in terms of cpio and emio. In the figure, We show the results

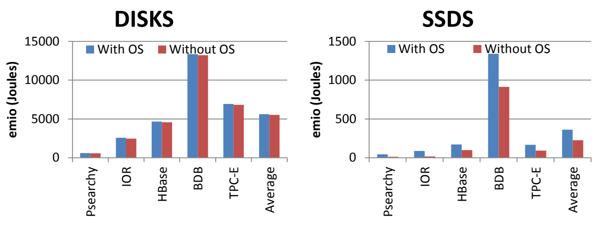

Figure 3: Energy Overhead of the System Layer on DISKS and SSDS.

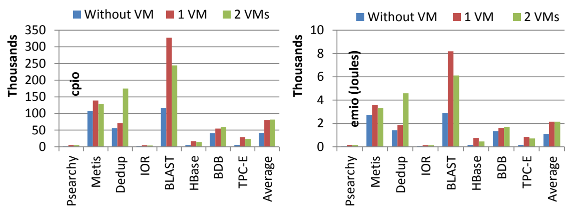

Figure 4: cpio (Left) and emio (right) with and without virtualization on SSDS.

with one instance of workload without virtualization, with the same workload running inside a VM and two instances of workload each in a separate VM. In terms of performance, we see thati, on average, cpio increases by 150%. Note that three workloads including Dedup, Metis and BDB have low overhead up to 30% due to virtualization compared to other workloads. Note that the percentage of system time out of total execution time for the three workloads is 0.6%, 7.2%, and 10% respectively. There is a possibility that these application operate mostly from the caches maintained by the application or the host OS. For other workloads, the overhead due to virtualization is upto 300%.

For most workloads, there is some reduction in virtualization overhead when two VMs are used to run two instances of the same workload. In particular for IOR and BLAST, there is a 16% and 25% reduction in cpio when two VM instances are run compared with running one VM instance. However, we note that for Dedup and BDB, the overhead due to running two VMs is actually increased. In particular for Dedup, there is a 200% increase in VM overhead with two VMs compared to only 28% overhead with one VM.

Finally, we note a similar trend in terms of energy overheads of virtualization. On average for all workloads, virtualization results in an increase in energy consumption of 180%. The overhead is particularly adverse for two important workloads for today's data-centres including HBase and TPC-E. For the two workloads, the overhead is 350% and 400% respectively.

7. SCALABILITY OF OS LAYER

Ideally, with increasing number of cores, application should perform proportionally more I/O thus resulting in constant cpio. However, we observe that this is not true with today's system stacks. In this work, we are mainly interested in the system component of cpio. We show the increase in system cycles per I/O operation with the number of cores

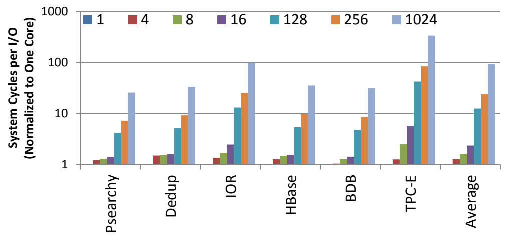

Figure 5: Increase in System Cycles per I/O Operation from 1 to many cores.

in Figure 5. We show the measured values for 1, 4, 8, and 16 cores[1]. We then project the results for up to 1000 cores using a linear model. We show the increase in system cycles normalized to the measured cycles with one core.

System cycles increase for all applications shown in Figure 5. The same applications running with a many-core processor with 1024 cores will, on average, spend 90x more system cycles per I/O operation. This indicates that the current system layer today is inefficient and improvements are needed to scale applications that heavily use the system layer. We note that, in particular for the more recent OLTP workload, the increase in system cycles with the number of cores is dramatic. TPC-E relies heavily on the OS services, spending up to 36% of its execution time in the OS with 16 cores.

The system cycles consumed by an application translates directly to energy overheads. Thus, the non-scalability of the OS layer has adverse implications for energy consumption in data-centres.

8. RELATED WORK

Interest in research on issues related to inefficiency of software stacks in data-centeric applications is increasing. [17] discusses trends in building datacenter applications from existing components that lead to large inefficiencies. Recent work has been pointing out that software stack inefficiencies have an impact on the energy efficiency of datacenter infrastructures [3, 14].

The dominant methodology so far on the power reduction side has been to construct power models by first taking measurements for a subset of the design space. Using this approach, there is strong evidence by [9] and [20] that directly relates the power consumption in clusters running typical data centric workloads to the CPU utilization and physical memory usage. The measurements in [9] further suggest that CPU and physical memory are major contributors to power. The actual modeling techniques are developed and discussed extensivly in [19, 18, 8].

The ENERGY STAR program of the U.S. Environmental Protection Agency estimates the infrastructure requirements and energy consumption of data-centres in 2020 based on market growth [1]. They also suggest optimizations that will likely reduce energy budgets mainly by reducing the power of single server machine used in data centres.

[1]For applications used in this work, we observe a similar scalability trend both with increasing number of cores and increasing number of hardware threads. Thus, the measured value shown for 16 cores is with 8 cores and hyper-threading enabled.

There has been work in the past on analyzing the performance of modern Operating System layers. More recently, [5] identified many bottlenecks in the Linux kernel when scaling to many cores. A similar scalability analysis is decribed for many important application domains by [7, 21]. Finally, there has been recent work in reducing overheads due to the addition of virtualization layer for I/O intensive workloads. The authors in [13] propose Split-X that optimizes the guest-host OS crossings, whereas [11] proposes ELI to optimize the interrupt path under VMs.

9. CONCLUSIONS

In this work we use a set of I/O intensive applications to quantify the overhead of the OS and virtual machines when performing I/O relative to application processing. We find that the OS can result in an increase in energy consumption of 60%. The addition of VMs results in an additional energy overhead per I/O operation of 150%. Finally, by looking at an increasing number of cores we find that per I/O overhead in the OS increases by up to 90x when projecting trends to 1000 cores using a simple linear model.

10. ACKNOWLEDGMENTS

We thankfully acknowledge the support of the European Commission under the 7th Framework Programs through the IOLANES (FP7-ICT-248615), HiPEAC2 (FP7-ICT-217068), and SCALUS (FP7-PEOPLE-ITN-2008-238808) projects.

11. REFERENCES

[1] U. E. P. Agency. Report to congress on server and data center energy efficiency. www.energystar.gov.

[2] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[3] E. Anderson and J. Tucek. Efficiency matters! *SIGOPS Oper. Syst. Rev.*, 44(1):40–45, 2010.

[4] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, PACT '08, pages 72–81, New York, NY, USA, 2008. ACM.

[5] S. Boyd-Wickizer, A. T. Clements, Y. Mao, A. Pesterev, M. F. Kaashoek, R. Morris, and N. Zeldovich. An analysis of linux scalability to many cores. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.

[6] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 143–154, New York, NY, USA, 2010. ACM.

[7] Y. Cui, Y. Chen, and Y. Shi. Scaling oltp applications on commodity multi-core platforms. In *Performance Analysis of Systems Software (ISPASS), 2010 IEEE International Symposium on*, pages 134 –143, march 2010.

[8] D. Economou, S. Rivoire, and C. Kozyrakis. Full-system power analysis and modeling for server environments. In *In Workshop on Modeling Benchmarking and Simulation (MOBS*, 2006.

[9] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, pages 13–23, New York, NY, USA, 2007. ACM.

[10] J. F. Gantz, D. Reinsel, C. Chute, W. Schlichting, J. Mcarthur, S. Minton, I. Xheneti, A. Toncheva, and A. Manfrediz. IDC - The Expanding Digital Universe: A Forecast of Worldwide Information Growth Through 2010 (updated in 2010, 2011)., Mar. 2007.

[11] A. Gordon, N. Amit, N. Har'El, M. Ben-Yehuda, A. Landau, D. Tsafrir, and A. Schuster. Eli: Bare-metal performance for i/o virtualization. 2012.

[12] R. Joshi. Data-Centric Architecture: A Model for the Era of Big Data. http://drdobbs.com/web-development/229301018 .

[13] A. Landau, M. Ben-Yehuda, and A. Gordon. Splitx: split guest/hypervisor execution on multi-core. In *Proceedings of the 3rd conference on I/O virtualization*, WIOV'11, pages 1–1, Berkeley, CA, USA, 2011. USENIX Association.

[14] J. Leverich and C. Kozyrakis. On the energy (in)efficiency of hadoop clusters. *SIGOPS Oper. Syst. Rev.*, 44(1):61–65, 2010.

[15] J. Li, B. T. Loo, J. M. Hellerstein, M. F. Kaashoek, and et al. On the feasibility of peer-to-peer web indexing and search. In *IN IPTPS.03*, pages 207–215, 2003.

[16] llnl.gov. ASC Sequoia Benchmark Codes. https://asc.llnl.gov.

[17] N. Mitchell. The big pileup. In *Performance Analysis of Systems Software (ISPASS), 2010 IEEE International Symposium on*, pages 1 –1, mar. 2010.

[18] S. Rivoire, P. Ranganathan, and C. Kozyrakis. A comparison of high-level full-system power models. In *HotPower'08: Proceedings of the 2008 conference on Power aware computing and systems*, pages 3–3, Berkeley, CA, USA, 2008. USENIX Association.

[19] S. Rivoire, P. Ranganathan, and C. Kozyrakis. A comparison of high-level full-system power models. In *HotPower'08: Proceedings of the 2008 conference on Power aware computing and systems*, pages 3–3, Berkeley, CA, USA, 2008. USENIX Association.

[20] A. Vasan, A. Sivasubramaniam, V. Shimpi, T. Sivabalan, and R. Subbiah. Worth their watts? - an empirical study of datacenter servers. pages 1 –10, jan. 2010.

[21] B. Veal and A. Foong. Performance scalability of a multi-core web server. In *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems*, ANCS '07, pages 57–66, New York, NY, USA, 2007. ACM.