# Understanding scalability and performance requirements of I/O intensive applications on future multicore servers

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

## I. Abstract

Today, there is increased interest in understanding the impact of *data-centric* applications on compute and storage infrastructures as datasets are projected to grow dramatically. In this paper, we examine the storage I/O behavior of twelve data-centric applications as the number of cores per server grows. We configure these applications with realistic datasets and examine configuration points where they perform significant amounts of I/O. We propose using *cycles per I/O* (cpio) as a metric for abstracting many I/O subsystem configuration details. We analyze specific architectural issues pertaining to data-centric applications including the usefulness of hyper-threading, sensitivity to memory bandwidth, and the potential impact of disruptive storage technologies.

Our results show that today's data-centric applications are not able to scale with the number of cores: moving from one to eight cores, results in 0% to 400% more cycles per I/O operation. These applications can achieve much of their performance with only 50% of the memory bandwidth available on modern processors. Hyper-threading is extremely effective and applications perform with in 15% of what is achieved with full cores instead of hyper-threading. Further, DRAM-type persistent memory has the potential to solve scalability bottlenecks by reducing or eliminating idle and I/O completion periods and improving server utilization. Projecting in the future for an increasing numbers of cores, at 4096 processors, servers would require between 250-500 GBytes/s per server and we will need about 2.5M servers that will consume 24 BKWh of energy to do a single pass over the projected 35 Zeta Bytes of data around 2020.

## II. Introduction and Motivation

Recently, there has been increased interest in examining how modern data-centric infrastructures will cope with the alarming rate of data growth; Published reports [1] evaluate the trends in data growth and show that by 2020 we will need to cope with 35 Zeta Bytes (ZB) of stored information. This growth is mainly driven by our ability to generate and collect more information at all levels of human activity and by new business models and services. In this context, there are increasing concerns that existing applications and infrastructures will not be able to cope with this growth in data, limiting our ability to process available information. These trends are represented by *data-centric* applications that are currently driving the shift towards the design of systems that process data rather than perform computation on memory resident datasets [2]. A particular concern when designing such systems is the efficiency and scalability of the I/O stack due to the I/O intensive nature of most data-centric applications.

For this reason, there has been work in understanding how our data processing infrastructures will cope with data as well as how they are projected to scale in the future, mainly with processor technology and the number of cores. The predominant approach for scaling modern applications to many cores is to improve our current system and software stacks for handling parallelism better. Wickizer et al. [3] report many scalability bottlenecks in today's system stacks. However, this approach of scaling the underlying system to larger numbers of cores is not straight-forward. For this reason, another possible approach, as advocated e.g. by Salomie et al. [4], is to "distribute" multiple instances of the same application as a means of utilizing additional cores.

To date, little attention has been paid to how I/O activity in applications scales with the number of cores and the resulting implications on data-centric applications and infrastructures. For instance, results reported in [3] consider only I/O to an in-memory filesystem. Moreover, lately there has been increased interest and work on projecting I/O requirements to the 2020 timeframe [5], [6]. These projections mainly aim to identify I/O requirements at the infrastructure and server-level and estimate potential limitations or set design targets. However, existing studies use market growth rate for storage and processing capability to project storage I/O requirements of servers and infrastructures making it difficult to examine the impact of specific optimizations and techniques.

In this work we tackle these issues by examining twelve applications that operate on data. We perform extensive measurements for calculating and understanding cpio for these applications that are currently used for **s**ervicing, **p**rofiling, **a**nalyzing and **m**anaging **d**ata (SPAMD). We start from measurements on servers representative of most modern data-centric infrastructures. We ensure that applications are operating at configuration points where they perform significant amount of I/O operations while generating maximum load for the processor. This is not a straight-forward step as most applications require extensive tuning or modifications to reach this state. Further, it requires extensive effort to tune the runtime parameters and collect datasets that are representative of

future workloads. We characterize each application and present an analysis of its behavior with respect to I/O.

We first examine data-centric applications scale with the number of cores from an I/O point of view. We examine the impact of hardware multi-threading, sensitivity to memory bandwidth, and the impact of memory-based persistent I/O. We propose using *cycles per I/O (cpio)* as an overall metric that abstracts system-level details and is able to capture application behavior. We discuss how we measure cpio and then we use it to investigate the issues above by performing measurements on a real system with realistic workloads. We focus on the following questions:

- How does the amount of I/O performed by applications increases with the number of cores and what happens to the number of cycles per I/O?
- Should processors targeted for data-centric applications support hyper-threading?
- What is the impact of memory bandwidth on application behavior?
- To what extent can "storage-class-memories" improve the scalability of modern applications to many cores?

Then, we perform a range of projections for infrastructure size and server I/O requirements as datasets and the number of cores per server grows in the 2020-timeframe. We use our measurements and a simple linear model to set targets for I/O requirements in future servers and to also provide a first approximation of the infrastructure size that will be required to process the data produced in 2020. We focus on the following questions:

- What I/O requirements, in terms of throughput and IOPS, will a server have as the number of cores grows?
- What infrastructure size in terms of the number of servers/cores and energy will be required to process data in the 2020-timeframe?

We compare our projections using cpio with estimates from market growth and find that the two approaches result in compatible numbers. However, unlike market growth information, our approach using cpio allows for projecting the impact of new designs and techniques on application and infrastructure I/O efficiency.

Overall, our work is a step towards building a methodology for evaluating and understanding the impact of architectural and systems software optimizations and extensions on data-centric applications and infrastructures. Our results indicate that:

- Compared to one core, applications spend from 0% up to 400% more cycles per I/O with 8 cores.
- On average, for twelve data-centric workloads, there is only a 15% increase in cycles per I/O when hyper-threading is used instead of full cores.
- On average, for seven data-centric workloads, there is only a 20% increase in cycles per I/O, when memory bandwidth per core is reduced by 37%.
- Persistent storage operating at the speed of DRAM is able to fully eliminate periods from the execution time where applications merely wait for I/Os to complete. Idle periods, although not fully, but to a great extent are eliminated from the execution time.
- If applications scale according to our optimistic assumptions, servers with 4096 cores will need to support storage I/O throughput between 250-500 GB/s.
- A one pass over the entire dataset that will be produced in 2020 will require 18 billion kWh given current server power, idle power, and scalability trends. This energy will be consumed by 2.5 million servers operating for one year at low utilization levels observed today. We show that, whereas architectural improvements such as reducing idle power and minimizing total server power will result in up to 2x saving in energy consumption, scaling applications to reduce per I/O overhead has the potential to reduce energy consumption by a factor of 100x.

The rest of this paper is structured as follows. In Section III, we describe a methodology for characterizing and projecting I/O behavior. Section IV describes a set of applications that operate with big datasets. Section V discusses our experimental platforms and observed behavior of applications. Section VI provides measurements from experiments and Section VII provides projections for future. Finally, Section VIII discusses related work and Section IX concludes the work.

## III. Methodology

In this section we describe our methodology for characterizing I/O behavior based on cycles per I/O.

### A. Using cpio to characterize data-centric applications

Characterizing I/O is a complicated process. This is particularly true today, with storage systems that include many layers, components, and perform complex functions transparently to the applications. For this reason it is important to use metrics that can abstract the details of today's storage subsystems and their interaction with applications. We propose using *cycles per I/O operation* (*cpio* ) as a metric to quantify application behavior from an I/O point of view. We also propose to use *cpio* for projecting the scalability and requirements of data-centric applications.

We calculate *cpio* for each application by running each application in a *meaningful* configuration; applications when run, should generate I/O traffic. For instance, cases where the workload fits in the available memory and exhibit low I/O are probably not typical of future configurations since the demand for data grows faster than DRAM capacity. For this purpose, we select datasets that are big enough to not fit in memory and generate I/O throughout execution.

We measure the average execution time breakdown as reported by the OS consisting of user, system, idle, and wait time. Linux reports the CPU utilization numbers in "USER_HZ" which is 100 ms on our systems. The conversion from USER_HZ to physical cycles is straightforward given the frequency of each core is known. We also note the number of I/Os (NIOS) that occurred during the same interval counting both read and write operations. There are two issues related

to the *cpio* calculation. First, what each of the components means and second which ones should be taken into account to come up with a meaningful metric. We next briefly explain what each component of the breakdown means.

*User time* refers to the time an application spends executing code in the user space. When the user application request services by the OS, the time spent is classified as *system time*. The time an application spends waiting for I/Os to complete is classified as *iowait time*. *Idle time* refers to the time during which the application either has no more work to perform within its allocated quantum or because it is waiting for resources that are not available, for instance, locks.

Given this, *cpio* can be calculated as follows: Either divide the total number of cycles with the number of I/Os including idle and iowait time or by excluding these two components. The first approach is valid for cases where it is difficult to scale the application and consume all available CPU cycles. This is common today given the complexity of applications and the increasing number of cores. This first approach is also more representative of systems where idle cycles cost significantly (disproportionally) in terms of energy which is the situation today [7]. However, given that one of our goals is to quantify issues in a longer timeframe and that there is currently significant research interest in providing energy proportionality in future systems, we choose to exclude idle and wait cycles from the *cpio* calculation, assuming that these cycles have no cost.

A second issue is what should be used as the *number of I/Os*. Typically in I/O intensive applications, there are multiple parameters that can characterize I/O behavior: the size of each I/O (e.g. small vs. large), the type of I/O (read or write), the access pattern (e.g. sequential vs. random), and the number of outstanding I/Os. In many cases it is difficult or impossible to capture this behavior and associate this to application behavior. In our work, we use a fixed and predefined size of 512 bytes as the number of I/Os that occur during execution. We choose the size to be 512 bytes which is a typical size for the sectors in many (but not all) storage devices. The main implication is that the overhead per I/O in an application that does large I/Os will appear to be smaller per I/O. We consider this to be a desirable situation, since an application (or system) that strives to achieve large I/Os should be perceived as a good case. Finally, note that the CPU utilization and I/O numbers we refer to are cumulative for the whole system and the execution window we consider regardless of the number of cores or different phases in the application.

Thus, *cpio* is calculated by dividing the user plus system cycles as reported by the OS with the total number of sectors read and written throughout the execution of application. In all our measurements, we run applications long enough to capture their typical behavior.

The advantages of using *cpio* as a metric to examine data-centric applications are: 1) Application-level metrics, such as transactions or events do not represent the actual volume of data that pass through the I/O subsystem. *cpio* , on the other hand, takes both compute and I/O resources consumed into account. 2) Individual components of *cpio* are able to point towards inefficiencies in different components of today's complex software stacks. For instance, the ability to quantify user-level versus system-level overheads. 3) Given the peak and idle power of a server, *cpio* can be converted to energy consumed per I/O operation. 4) When running multiple applications each with its performance metric, it is difficult to analyze the overall system performance. *cpio* serves as a system-wide metric of efficiency in such scenarios. 5) *cpio* has predictive value and is able to project future infrastructure requirements as discussed in the next section.

### B. Using cpio as the basis for projections

We use *cpio* to project application requirements, as follows. A multi-core processor with $N$ cores, each with a frequency $F$, has a total capacity of $C_n$ physical cycles during an interval $T$, given by $C_n = N * F * T$. We define the CPU utilization, $\mu$, as $C_p / C_n$, where $C_p$is the sum of user and system cycles during execution. $NIOS$ is the total number of 512-byte I/Os performed by an application during an interval $T$. Using the above, *cpio* can be expressed as:

$$cpio = (\mu * N * F * T)/NIOS. \quad (1)$$

To calculate IOPS that I/O subsystems will need to support with increasing number of cores, we use the following equation:

$$IOPS = (\mu * N * F)/cpio. \quad (2)$$

Note that, in Equation 1, $IOPS = NIOS/T$.

We assume a frequency of 2 GHz for each core in future servers, opting for the current trend of a larger number of slower cores. To estimate the size of infrastructures in data-centres we use Equation 2 to find throughput of applications. We then assume a particular size for the dataset and a time frame with in which to process the dataset. This provides us with the number of cores (or servers) required to process the projected amount of data. We then use various assumptions about the power of mid-range servers in 2020 to estimate energy consumption of all the servers.

## IV. SPAMD Applications

In this section we discuss how we configure the applications we use to perform large amounts of I/O and the datasets we use. Table I summarizes these characteristics.

*Asynchronous direct I/O (zmIO)* is an in-house micro-benchmark able to provide data rates close to maximum storage bandwidth, using the asynchronous I/O API of the Linux kernel to issue concurrent I/Os at low CPU utilization. It serves as a reference point quantifying the maximum achievable performance in today's systems [8].

*File system stress* tests can reveal the behaviour of metadata-intensive workloads. We use a modified version of fs_mark [9] to stress filesystem operations individually. Each fs_mark thread performs a sequence of operations on a private file within a shared directory.

*Checkpointing* of high performance computing (HPC) applications generates large amounts of I/O. We use IOR [10]

TABLE I
**SPAMD**: APPLICATIONS FOR **S**ERVICING, **P**ROFILING, **A**NALYZING AND **M**ANAGING **D**ATA.

| Application | Description | Parameters | Dataset Size (GBytes) |
|---|---|---|---|
| zmIO | Asynchronous Direct I/O | Outstanding I/Os=16K; Read/Write Block Size=128KB | 600 |
| fs_mark | File System Stress Test | Threads=128; Directories=128; Files per Directory=128<br>File Size=4MB; Read/Write Block Size=16KB<br>Operations=open,create,read,write,close | 64 |
| IOR | Application Checkpointing | Processes=128; File Size=2GB; I/O Mode=MPI_IO;<br>Offseting within File=Sequential | 128 |
| Psearchy | File Indexing | H(Directory Hierarchy)=Flat(F) or Complex(C)<br>D(Document Size)=Small(S) or Large(L)<br>Workloads: I-HFDL; I-HFDS; I-HTDS; I-HTDL<br>Processes=32; Hash Table Size=128MB | 100 |
| Dedup | File Compression | DedupS: File Size=10MB;Instances=100; Threads per Stage=1<br>DedupL: File Size=1GB; Instances=10; Threads per Stage=32 | 1<br>10 |
| Metis | Mapreduce Library for<br>Single Multi-core Machines | Application:Word Count; Instances=5<br>File Size=1GB; Map Threads=8; Reduce Threads=8 | 20 |
| Borealis | Data Streaming | BR-128 : Tuple Size=128 Bytes; Batching=256; Instances=4<br>BR-1024 : Tuple Size=1KB; Batching=32 ; Instances=4<br>BR-64 : Tuple Size=64KB; Batching=1 ; Instances=4 | 2.62<br>2.62<br>60 |
| HBase | NoSQL Store | Threads=128; Fields per Record=10; Size of Field=1KB | 30 |
| BDB | Key-value Store (Java-based) | Threads=128; Fields per Record=10; Size of Field=1KB; | 30 |
| TPC-C | OLTP Workload (Warehouse) | Warehouses=3000; Virtual Users=512<br>innodb_thread_concurrency=8; innodb_file_io_threads=4 | 155 |
| TPC-E | OLTP Workload (Stock Broker) | Active Customers=200000; Days of Trade=7; Terminals=128<br>innodb_thread_concurrency=8; innodb_file_io_threads=4 | 155 |
| Ferret | Content Similarity Search | Size of Query Images=800KB | 200 |
| BLAST | Sequence Similarity Search | Instances=16; Threads per Instance=16; Task=*blastn*<br>Queries per Instance=16; Databases=Pre-formatted Nucleotide-<br>Databases from NCBI (refseq_genomic, env_nt, nt)<br>Alignments=128; Target Sequences=5000 | 20 |
| Tariff | Offline Profiling of<br>Call Detail Records (CDRs) | Threads=64; Size of each CDR File=1GB | 64 |

to simulate various checkpointing patterns. IOR uses MPI and typically exhibits moderate user time whereas the I/O issued by several concurrent MPI processes results in significant iowait time.

*File indexing* is mainly done as a back-end job in data-centres and web hosting facilities. We use Psearchy [3] as a file indexing application. We run Psearchy using multiple processes where each process picks files from a shared queue of file names. Each process maintains a hash table for storing BDB indices in memory. The hash table is flushed to storage devices after it reaches a particular size. We modify the original Psearchy to improve its I/O behavior by avoiding character I/O and batching requests to larger sizes.

*Deduplication* is a technique for compression mainly used for near-line and archival storage. We use the dedup kernel from the PARSEC benchmark suite [11]. Dedup is usually entirely dominated by user time.

*Mapreduce* is used as the basis for various data-centric applications. We use Metis from the Mosbench benchmark suite [3] and perform a simple wordcount on input files. Metis maps the input file in memory and assigns a portion of the file to each of the *map* threads. Metis is primarily dominated by user time. We run multiple instance of Metis and assign a different file to each instance.

*Stream processing* is an upcoming class of applications for data-centres. We use Borealis [12] to process streams of records (or tuples) stored on storage devices. We run the client, the Borealis server and the receiver on the same node. The client reads tuples from storage devices, the Borealis server filters tuples, and the receiver writes the tuples to storage devices. We extensively hand-tune Borealis to remove operations that hurt overall throughput. We run multiple instances of Borealis to increase system utilization. Two important parameters for Borealis are: tuple size and the batching factor. Batching factor is the number of tuples in an event. As batching factor is increased while keeping the tuple size small, there is an increase in user time.

*NoSQL data stores* are becoming popular for serving data in a scalable manner. HBase is such a data serving system that is part of the Hadoop framework. We use the YCSB benchmark [13]. 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 and 30% update 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 over traditional databases* is another

approach to building NoSQL stores. BDB is a library that provides support for building data stores based on key-value pairs. Our evaluation methodology for BDB is similar to that for HBase. Since BDB is an embedded data store, the YCSB clients and the BDB code share the same process address space. 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-C and TPC-E as OLTP workloads. TPC-C models an order-entry environment of a wholesale supplier while TPC-E models transactions that take place in a stock brokerage firm. We run both TPC-C and TPC-E using MySQL and specify runtime parameters that result in high concurrency. We use an open-source version of TPC-C called Hammerora [14]. We run the hammerora clients and the MySQL database server on the same machine. We observe that hammerora clients consume very little percentage of the entire CPU utilization in our experiments. We run TPC-C with 6GB RAM which results in realistic amount of I/O for our chosen database. We observe that using MySQL database server results in high idle time for both TPC-C and TPC-E.

*Content similarity search* is used in data-centres that host e.g. social networking services [11]. We use Ferret from the PARSEC benchmark suite. Ferret is compute intensive and performs sustained but small amount of I/O. We fit the database of image signatures against which queries are run in memory. Ferret is dominated by user time.

*Comparative genomics* leverages the tremendous amount of genomic data made possible by advances in sequencing technology. We use BLAST [15] 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 5KB, which is a common case in proteome/genome homology searches. BLAST is I/O intensive and the execution time is dominated by user time.

*Profiling of call detail records (CDRs)* by telecommunication service providers is performed for analyzing the feasibility of various usage plans. We use TariffAdvisor (Tariff) that does offline profiling of CDRs using machine learning models. The input to Tariff is a set of files that each contains different plans offered by the operator to the users. Tariff analyzes the records covering a period of time and outputs the plans that are financially productive. Tariff uses PostgreSQL as the database management system. Tariff is an I/O intensive application and its runtime is dominated by user time.

## V. Experimental platform

Figure 1 shows the breakdown of execution time on a disk-based and an ssd-based storage subsystem. The important features of the two machines are shown in Table II. The applications on the X-axis are ordered in terms of increasing iowait time, as a percentage of total execution time. Note that in terms of I/O behavior, the applications cover a broad range and the average iowait time reduces when using SSDs.

TABLE II
Summary of experimental platform 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 note that the average idle time for DISKS is 30% compared to only 7% for SSDS. This is because, threads wait longer for I/Os to complete, and thus the entire system is slow. 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. We also note that on average, the system time on DISKS is only 2% of total execution time compared with 26% on SSDS. Thus, the idle time that applications observe on DISKS is converted to system time on SSDS for the applications that we evaluate.

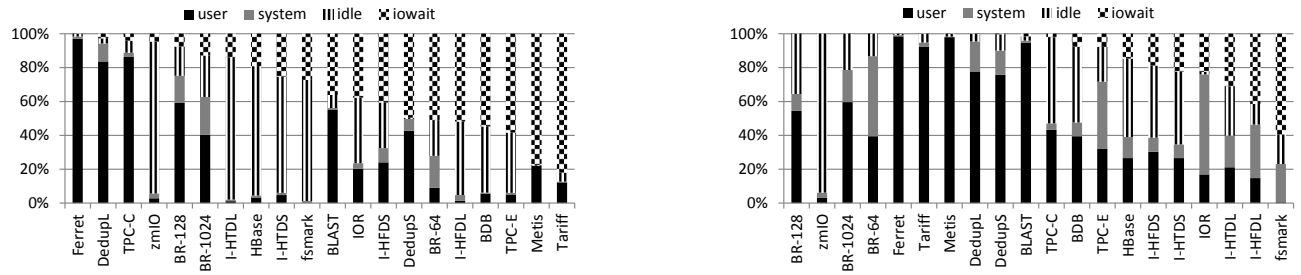
Figure 2 shows the cpio of all applications for both setups. We observe that for many applications, cpio is largely independent of the underlying storage technology and system configuration. Figure 2 does not show the cpio of Ferret and TPC-C since they would require a much higher value on the Y-axis. The cpio of Ferret is approximately 2 million cycles on both setups. The cpio of TPC-C on SSDS and DISKS is 12K cycles and 2M cycles respectively. We suspect this large difference is due to spinning by client threads which consume excessive cycles on SSDS which has more cores.

Note that given two applications, the one with a higher execution time can appear to be more cpio-efficient by doing large amounts of I/O. As an example, note that BDB, which is a light-weight data store, has an order of magnitude higher cpio compared to HBase, which has a complex software stack. This is because HBase does a large amount of small I/O operations. In reality, its execution time for the same YCSB test is much higher compared to BDB. Thus, cpio should be used carefully when comparing the efficiency of two software stacks that generate different volume of I/O during execution.

## VI. Measured results

### A. Does application I/O scale?

Perfect scalability for I/O intensive applications implies that, with doubling of cores, IOPS should proportionally double. Thus, cpio stays constant and application performance doubles. However, we show in Figure 3 that cpio increases for most applications from one to four cores and for all applications from one to eight cores. In the same figure, we show scalability of cpio with hardware threads instead of




(a) DISKS (b) SSDS

Fig. 1. Breakdown of execution time in terms of user, system, idle, and iowait time on DISKS and SSDS.

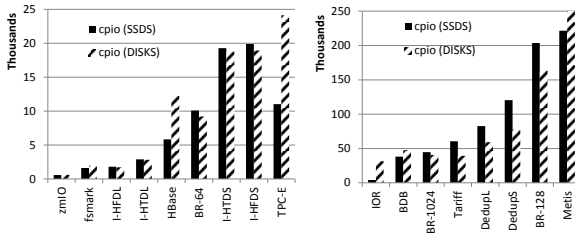
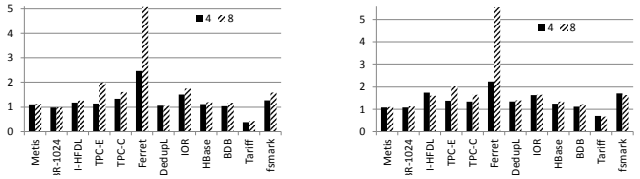



Fig. 2. cpio on DISKS and SSDS.




(a) Cores (b) Hardware Threads

Fig. 3. Increase in cpio (Y-axis) normalized to the cpio with one core (a) and with one hardware thread (b).

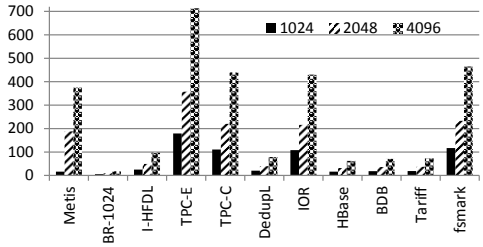



Fig. 4. Increase in cpio (Y-axis) from 1 to many cores normalized to cpio with one core.

cores. In Figure 3(b), we use one and two cores per socket on SSDS and enable hyper-threading to experiment with four and eight hardware threads. We note that the scalability trend is the same with increasing hardware threads as seen with increasing cores.

Next we show how well applications are able to use cycles made available by more cores for processing additional I/Os. Figure 5 shows how $\mu$ changes from one to multiple cores. From one to eight cores, $\mu$ drops for most applications, and up to 0.68 from one. Further, from one to 16 hardware threads, $\mu$ drops to below 0.5 for HBase, BDB, TPC-C and I-HFDL. This drop is because as more cores are added, either iowait and/or idle time increases. Thus, as applications strive to perform more I/O operations with increasing number of cores, synchronization overhead becomes one of the primary bottlenecks to scalability. For other workloads, in particular Dedup, Metis, Tariff, and BR-1024, $\mu$ does not drop significantly.

### *B. Are hyper-threads effective?*

In this subsection we show the effectiveness of hyper-threading for data-centric applications. Schone et al., recently showed the (slightly) negative impact of hyper-threading on performance [16]. However, they experimented with all cores and hyper-threading enabled. With all cores utilized, it is difficult to analyze if any particular feature is the bottleneck. Therefore, we evaluate hyper-threading with different number of cores. Figure 6 shows cpio for different number of cores both with and without hyper-threading enable (normalized to cpio with four cores). We note that, for most applications, there is no significant increase in cpio using four cores with hyper-threading (4C+HT) instead of eight cores (8C). In particular, most applications observe only a 20% increase in cpio when hardware threads are used instead of full cores thus achieving performance within 80% of performance with full core.

This figure also shows that, for half of the workloads, cpio increases significantly with 16 hardware threads. Given our earlier observation that cores and hardware threads follow a similar scalability trend, we believe that, what we observed for 16 hardware threads, will be the case for 16 cores. Our results indicate that the increase in cpio is contributed both by the user and system component. Thus, we infer that the I/O stack in current systems do not scale because of resource contention for shared resources, for instance, a single page cache shared across all software threads.

### *C. How much memory bandwidth?*

An important question for data-centric applications is how much memory bandwidth is sufficient for scalability to many cores. We answer this question by analyzing the sensitivity of applications to memory bandwidth. Scientific applications are known to be less sensitive to memory bandwidth, because computing complex addresses generated by these applications hides the memory latency [17]. Since it is difficult to estimate

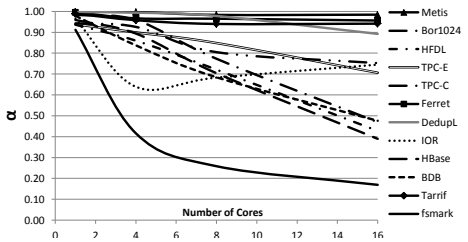



Fig. 5. Drop in CPU utilization ($\mu$) from 1 to 16 cores.

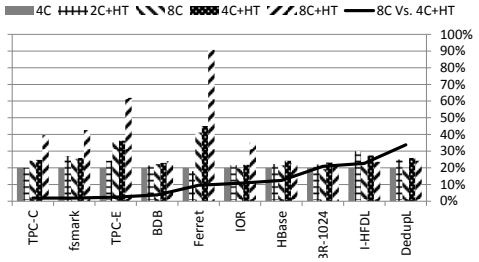



Fig. 6. Effectiveness of hyper-threading for data-centric applications.

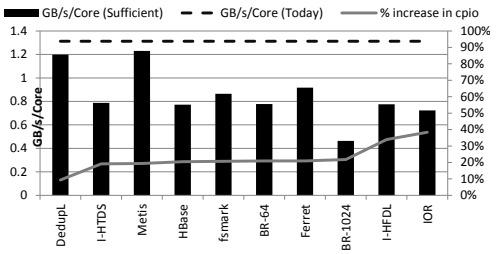



Fig. 7. Memory bandwidth per core today and after tolerating an increase in cpio.

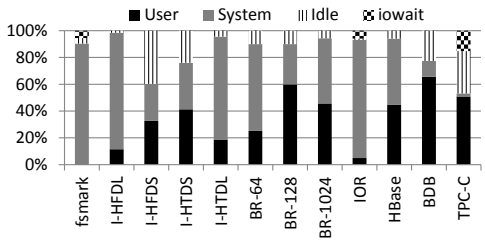



Fig. 8. Impact of DRAM-based storage on application behavior.

what future technologies might be capable of providing, we project memory bandwidth demand in future that is sufficient for scaling current performance standards to many cores. First, the maximum bandwidth on SSDS is 21 GB/s. This implies a bandwidth of 1.3125 GB/s per core. We measure the increase in cpio by a decrease in memory bandwidth. For this, we wrote a microbenchmark modeled after STREAM [18] called *mstress* that stresses the memory subsystem. We run multiple instances of mstress along with an application from Table I. We note the aggregate memory bandwidth consumed by the mstress instances. Figure 7 shows the percentage increase in cpio of the application when part of the memory bandwidth is taken by mstress. Note that most applications suffer a 20% increase in cpio but then require from 6% up to 65% less memory bandwidth.

### D. What will be the impact of DRAM-type persistent storage?

An emerging challenge in the storage domain is to examine how things might evolve when storage class memories [5] start to appear in real systems. Although there are various types of memories proposed in this category, we make a first order approximation and take the simplistic approach that these memories will appear comparable to DRAM. However, we assume that applications will still perform I/O via the traditional I/O path, since this would be a first step in the evolution towards using storage class memories and in addition, complex data-centric applications such as transactional databases will require fundamental changes to avoid the traditional I/O path when going to persistent memory. For this study, we wrote a kernel module (kram) that simulates a block device. kram completes I/O operations in-place without using any re-scheduling of I/Os. The size of kram is 54 GB and the physical memory in these experiments is 12 GB. The experiments are performed on DISKS and datasets are adjusted to fit in the available memory.

Figure 8 shows the breakdown of execution time for selected applications with high iowait and idle times on SSDS. First, we see that iowait time disappears for all applications. This is expected since I/Os now complete upon issue without the issuing thread leaving the executing core. We note that, for certain applications, there is still idle time either due to severe imbalances or other form of synchronization. Our observations on idle time with DRAM-storage strongly indicate that application scaling on many-core systems will not be simple even for data-centric applications that in principle exhibit large amounts of concurrency.

## VII. Projected results

In this section, we use cpio to project I/O-related requirements consumption to the 2020 time-frame. In our results, wherever we show averages for all applications, we do not include zmIO, fsmark, and Ferret.

### A. How many cycles per I/O?

Figure 4 shows the projected increase in cpio for 1024, 2048 and 4096 cores using the measured values today. Note that, workloads such as I-HFDL, which observe only a small overhead in cpio with eight cores will end up spending 100 times more cycles per I/O with 4096 cores. This is an important observation given that future systems and servers that will process data are expected to have a large number of cores in a single enclosure.

### B. How much storage bandwidth?

We calculate millions of IOPS (MIOPS) using Equation 2 and use it to examine how much I/O we will need to provide

TABLE III
MILLIONS OF IOPS (MIOPS) REQUIRED FOR DATA-CENTRIC APPLICATIONS WITH 4096 CORES.

| Scenario | Millions of IOPS with 4096 Cores | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $s(\mu, cpio)$ | zmIO | fsmark | Ferret | I-HFDL | I-HFDS | I-HTDS | I-HTDL | Metis | DedupS | DedupL | BR-64 | BR-128 | BR-1024 | IOR | Tariff | BLAST | HBase | BDB | TPC-C | TPC-E | Average |
| $s(l,p)$ | - | 4.56 | 0.02 | 12 | - | 0.67 | - | 1 | 0.60 | 5 | 2 | 0.4 | 8 | 35 | 5 | - | 12 | 2 | 6 | 9 | 7.5 |
| $s(h,p)$ | - | 92 | 0.02 | 65 | - | 424 | - | 1 | 0.87 | 8 | 40 | 40 | 38 | 49 | 5 | - | 107 | 12 | 16 | 14 | 59 |
| $s(l,t)$ | 1228 | 1272 | 3 | 2207 | 163 | 151 | 1183 | 36 | 61 | 95 | 711 | 26 | 145 | 1522 | 128 | 147 | 563 | 103 | 321 | 535 | 476 |
| $s(h,t)$ | 14297 | 5038 | 3 | 4509 | 412 | 425 | 2822 | 37 | 68 | 99 | 812 | 40 | 183 | 1991 | 135 | 153 | 1405 | 215 | 680 | 743 | 818 |
| $s(l,d)$ | - | 1297 | 27 | 2540 | - | 146 | - | 79 | 74 | 132 | 811 | 25 | 140 | 3735 | 87 | - | 644 | 104 | 797 | 1743 | 969 |
| $s(h,d)$ | - | 7629 | 28 | 5941 | - | 405 | - | 80 | 83 | 148 | 989 | 38 | 186 | 5014 | 93 | - | 1652 | 218 | 1699 | 2469 | 1810 |

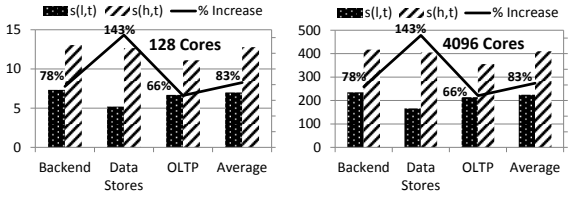



Fig. 9. GB/s (Y-axis) required by OLTP, Data Stores, and Backend applications in 2020. The solid line shows % increase in storage I/O from $s(l,t)$ to $s(h,t)$.

per server when running data-centric applications on many-core processors. Note that in Equation 2, $\mu$ and cpio, as a combination, leads to various scenarios, which we call $s(\mu, cpio)$. In this work, we consider $s(l,p)$, $s(h,p)$, $s(l,t)$, $s(h,t)$, $s(h,d)$, $s(l,d)$; where $l$ and $h$ stands for low and high utilization respectively whereas $t$, $p$ and $d$ stands respectively for cpio measured today with 16 cores, projected cpio from real measurements and desired cpio measured with one core.

We use $100\%$ for calculations with high utilization. For low utilization, we take the measured utilization with 16 cores for $s(l,t)$ and project it to many cores for $s(l,p)$. Note again that utilization could be low for several reasons including I/O bottlenecks and excessive synchronization. Thus, it is important to quantify all scenarios that may appear in future. The variation in server utilization is also observed in today's data-centres: The peak levels are characterized by high utilization, while mostly, servers operate at low levels of utilization [19].

We consider three values for cpio for a many-core processor in 2020. The measured value with 1 core is the most optimistic assumption for a many-core processor as it is challenging to completely eradicate contention and synchronization overheads. On the other hand, projecting from measured values is a very pessimistic scenario. Therefore, the measured value with 16 cores provides an interesting middle-ground between the two extremes.

Table III shows the MIOPS per server for the four scenarios assuming applications run on a single server with $4096$ cores. $s(l,p)$ results in the lowest MIOPS characterized both by low levels of server utilization and high cpio. $s(h,p)$ improves $\mu$ and thus results in increased IOPS by a factor of 4 (on average). This implies that even if cpio continues to increase, applications can exploit the large amounts of raw cycles to perform more I/O Operations per second. The more desirable scenarios result with measured cpio. Further, IOPS for $s(h,t)$ is, on average, twice the IOPS for $s(l,t)$ showing the importance of improving server utilization. Finally, using cpio with one core ($s(l,d)$ and $s(h,d)$) results in twice the IOPS compared to projected cpio.

There are workloads in Table III for which projected cpio becomes stable with increasing cores. For instance, note that BR-128 and I-HTDS generate the same IOPS for $s(h,t)$ and $s(h,p)$. These workloads observe utilization today between $40\%$-$50\%$. Still, note the adverse drop in throughput reported by $s(l,p)$ owing to increasing iowait time and idle time with more cores.

Figure 9 shows storage throughput in terms of GB/s for all workloads grouped into three categories. We show the throughput for a server with 128 cores and one with 4096 cores. We note that all workloads require 10s of GB/s with 128 cores assuming high utilization. However, for 4096 cores, the requirement is up to 500 GB/s for $s(h,t)$ and up to 250 GB/s for $s(l,t)$. With the current annual growth in disk throughput, this will require thousands of disks per server. Thus, faster and area-efficient storage devices will be required if modern applications are able to scale to many cores. Also, providing per socket memory and I/O bandwidth in the range of 100s of GB/s will be challenging. The servers in today's data-centres use network-attached storage via Ethernet or Fibre Channel links. Thus, packaging servers with enough network cards capable of providing multiple TB/s within expected power budgets will require tremendous efforts. Overall, improving application stacks for better CPU utilization and less overhead per I/O, will open up new and challenging problems in the storage and I/O domain.

### C. How much energy?

We show the energy that will be needed to sweep through (in one year) all data produced in 2020. We show the energy assuming both today's CPU utilization and full CPU utilization. We calculate energy considering idle power consumption in servers today (70% of full power) and assuming zero idle power. We use the measured, desired and projected cpio. We discard the case with projected cpio where CPU utilization is low since it is unrealistic to have a processor with 4096 cores which is mostly idle and applications spend 100x the cycles per I/O compared to what they spend on one core. We make different assumptions in Table IV for full power of a mid-range server in 2020: a) Energy-star based server power in 2006 (675W per server), which would correspond to about 0.1 Watt per core for a 4096 core server, if achievable, b) The same number projected to 2020 using Energy-star's projection (1.25 KW per server), which would correspond to about 0.2

TABLE IV
KWH (BILLIONS) TO SWEEP THROUGH 35 ZETA BYTES. IP=0% AND IP=70% REPRESENTS IDLE POWER OF 0% AND 70% OUT OF FULL SYSTEM POWER.

| Power Assumptions for a mid-range server in 2020. | Low CPU Utilization | | | | High CPU Utilization | | |
|---|---|---|---|---|---|---|---|
| | desired cpio | | today's cpio | | projected cpio | today's cpio | desired cpio |
| | IP=0% | IP=70% | IP=0% | IP=70% | | | |
| Assuming 0.5 Watts per core (2.5 kW) | 0.18 | 0.21 | 0.27 | 0.33 | 29 | 0.27 | 0.175 |
| Energy-star projected to 2020 (1.25 kW) | 0.11 | 0.12 | 0.16 | 0.20 | 17.5 | 0.16 | 0.107 |
| Energy-star of 2006 (0.675 kW) | 0.06 | 0.07 | 0.09 | 0.11 | 9.5 | 0.09 | 0.057 |

Watts per core, if achievable, c) 0.5 Watts per core (about 2.5-3.0 KW per server). We also assume that all power will be consumed by servers with data-centres operating at PUE=1.

Given today's utilization, projected cpio, idle power consumption, and power per server, it will take 13 Billion kWh to do one pass over data (on average). This represent $4.33\%$ total energy consumption in data-centres in 2020 projected by [20] based on annual growth. For a single pass over the dataset, this amount of energy consumption is extremely high. Therefore, even with machines built efficiently, where 4096 cores and supporting infrastructure such as I/O hubs and interconnect etc. is packaged in a 675 Watts server, application overheads will result in tremendous energy consumption in data-centres.

Bringing idle power down to zero has the potential to save energy consumption by up to $18\%$. Xiabo et al., [7] report a possibility of $50\%$ reduction in energy consumption of a real data-centre by bringing idle power down to $10\%$ of peak power. The difference in results is because of the application datasets and parameters we use, which result in an average CPU utilization of $68\%$, higher than what happens in today's data-centres.

Given packaging limitations, fitting 4096 cores in a single server (e.g. in four sockets) will require one to two orders of magnitude improvement from today's per-core power consumption to the 0.1 watt level. Our results in Table IV indicate that carefully architect in the software stack to avoid increasing cpio (as our measured trends show) can reduce energy consumption by one to two orders of magnitude. Finally, reducing idle power and improving server utilization are two other important factors, however with lower potential for energy consumption improvements, in the 10-30% range.

## VIII. RELATED WORK

In this section, we mainly compare our work with results reported in recent literature.

*a) Scalability to many cores:* Recently, there has been effort to scale Linux to multiple cores. The authors in [3] report and address scalability bottlenecks in all layers of the system stack. In [21] the authors analyze the scalability of many data-centric applications and provide a methodology to identify indiviual components of scalability bottlenecks. All these studies on the scalability analysis of data-centric applications use I/O to a memory-resident device or filesystem and focus on the CPU processing aspect alone [3], [22]. This approach potentially hides inefficiencies in applications that use a real storage subsystem, since they omit a number of layers in the OS kernel in addition to altering application behavior.

Low server utilization today is also an impediment when scaling to many cores. Figure 9 shows that for all application domains and data stores (HBase and BDB) in particular, there is a large gap between current utilization levels and what is ideally possible. Note that we use application parameters that result in high concurrency and little iowait time on the SSDS machine. Thus, it is worth discussing the argument of "distributing versus scaling" as presented in recent work, e.g. in [4]. It is a valid question how much effort will it take to improve utilization of a single instance on future many-core systems. Instead, it may be preferable to just run multiple instances of distributed applications on a single system, possibly via virtual machines.

*b) Storage I/O projections:* We contrast our projected throughput results with those reported in recent literature [5], [6]. Benner et al. [6] project the I/O bandwidth needed for OLTP, business intelligence, and technical applications in 2007 for today's time frame using historical trends. For the OLTP workloads, our observed storage I/O throughput with 16 cores match their results (approximately $1$ GB/s). Tariff and BLAST falls in the business intelligence and technical domains respectively. However, they perform storage I/O that is far from the projections of the authors. Note that application stacks in many domains tend to get complex over time increasing the cycles elapsed between successive I/Os that is not taken in to account by projections based on market growth.

Freitas and Wilcke [5] project storage I/O requirements of data-centric applications in 2020 using as a starting point, a large-scale system able to perform 2 million start I/O operations per second (SIO/s) in 2007. We observe, on average for all applications, roughly 2 million IOPS for 8 cores and then use four scenarios to project. Assuming $70\%$ and $90\%$ annual growth rate, the authors report 2 GSIO/s and 8.4 GSIO/s. Our most optimistic scenario i.e., $s(h,t)$ projects 1 billion IOPS per server assuming 4096 cores. As a reference point, zmIO reports 2 billion and 7 billion IOPS with 1024 and 4096 cores respectively.

*c) Energy projections:* Energy Star reports an 8x opportunity for saving energy consumption by improving power management of processors, voltage reduction, server consolidation using virtualization, etc. Note that Energy Star performs its projections by examining the growth rate for data-centre infrastructures over the last decade and taking into account power consumption projections for servers. Our calculations have a different starting point, which projects application behavior to many-core processors and reports implications of scaling on energy (in)efficiency in future data-centres. More recently, Ferdman et al. [22] analyze many data-centric applications and conclude that current processor architectures for servers are over-provisioned in terms of many microarchitectural features.

*d) cpio as an efficiency metric:* cpio as a characterization metric for data-centric applications is to some extent, similar to the five-minute rule by Jim Gray [23] and compute efficiency

metric discussed by Anderson and Tucek [24]. A performance comparison of different ways for managing disks in a virtualized environment by VMware [25] uses a metric called *MHz per I/Ops* which is similar to cpio. In this work, we show that cpio is able to characterize application behavior in terms of efficiency and scalability and it has predictive value.

## IX. Conclusions

In this work we first discuss the scaling of data-centric application to an increasing number as well as the impact of hyper-threading, memory bandwidth, and DRAM-type persistent memories to data-centric applications. Then, we discuss projections for I/O requirements in 2020 timeframe. We propose using *cycles per I/O* (cpio) as the basis for characterizing applications at a high-level and projecting requirements to the future and discuss implication of different approaches to calculate cpio.

Our measured results show that applications do not scale well with an increasing number of cores, despite the fact that they are designed to incur high concurrency. In addition, hyper-threading works very well for these applications, while today's servers are over-provisioned in terms of memory throughput. Finally, although DRAM-type persistent memory can help reduce idle time in applications, it is not enough to completely eliminate it.

Our projections show that future servers, under scaling assumptions, will need to support 250-500 GBytes/s of I/O throughput and that in 2020 we will need about 2.5M servers of 4096 cores and 24 BKWh of energy to do a single pass over the estimated 35 ZBytes of data that will be produced within 2020.

Overall, our methodology is able to abstract application behavior and to characterize a broad range of applications. For many cases, our methodology results in projections that are inline with published numbers, however, using measured data rather than market growth information. This makes our methodology useful as a tool to also evaluate the impact of system and architectural optimizations by examining the impact on $cpio$ .

## X. Acknowledgements

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. We are thankful to Neurocom LTD, Greece for providing us the code for TariffAdvisor and to Victor Molino and Martina Naughton for their feedback on the paper.

## References

[1] 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. [Online]. Available: http://www.emc.com/collateral/analyst-reports/expanding-digital-idc-white-paper.pdf

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

[3] 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*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–8. [Online]. Available: http://portal.acm.org/citation.cfm?id=1924943.1924944

[4] T.-I. Salomie, I. E. Subasu, J. Giceva, and G. Alonso, "Database engines on multicores, why parallelize when you can distribute?" in *Proceedings of the sixth conference on Computer systems*, ser. EuroSys '11. New York, NY, USA: ACM, 2011, pp. 17–30. [Online]. Available: http://doi.acm.org/10.1145/1966445.1966448

[5] R. F. Freitas and W. W. Wilcke, "Storage-class memory: the next storage system technology," *IBM J. Res. Dev.*, vol. 52, pp. 439–447, July 2008. [Online]. Available: http://dx.doi.org/10.1147/rd.524.0439

[6] A. Benner, P. Pepeljugoski, and R. Recio, "A roadmap to 100g ethernet at the enterprise data center," *Communications Magazine, IEEE*, vol. 45, no. 11, pp. 10 –17, november 2007.

[7] 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*. New York, NY, USA: ACM, 2007, pp. 13–23.

[8] "Kernel Asynchronous I/O (AIO) Support for Linux," http://lse.sourceforge.net/io/aio.html .

[9] R. Wheeler, "fs_mark," sourceforge.net/projects/fsmark/.

[10] llnl.gov, "ASC Sequoia Benchmark Codes," https://asc.llnl.gov.

[11] 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*, ser. PACT '08. New York, NY, USA: ACM, 2008, pp. 72–81. [Online]. Available: http://doi.acm.org/10.1145/1454115.1454128

[12] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. etintemel, Y. Xing, and S. Zdonik, "Scalable distributed stream processing," in *In CIDR*, 2003.

[13] 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*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 143–154. [Online]. Available: http://doi.acm.org/10.1145/1807128.1807152

[14] Steve Shaw, "hammerora," http://hammerora.sourceforge.net.

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

[16] R. Schone, D. Hackenberg, and D. Molka, "Simultaneous multithreading on x86_64 systems: an energy efficiency evaluation," in *Proceedings of the 4th Workshop on Power-Aware Computing and Systems*, ser. HotPower '11. New York, NY, USA: ACM, 2011, pp. 10:1–10:5. [Online]. Available: http://doi.acm.org/10.1145/2039252.2039262

[17] R. Murphy, "On the effects of memory latency and bandwidth on supercomputer application performance," in *Proceedings of the 2007 IEEE 10th International Symposium on Workload Characterization*, ser. IISWC '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 35–43.

[18] John D. McCalpin, "STREAM: Sustainable Memory Bandwidth in High Performance Computers," http://www.cs.virginia.edu/stream/ .

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

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

[21] S. Eyerman, K. D. Bois, and L. Eeckhout, "Speedup stacks: Identifying scaling bottlenecks in multi-threaded applications," in *ISPASS*, 2012.

[22] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the Clouds: A Study of Emerging Workloads on Modern Hardware," Tech. Rep., 2011.

[23] J. Gray and F. Putzolu, "The 5 minute rule for trading memory for disc accesses and the 10 byte rule for trading memory for cpu time," *SIGMOD Rec.*, vol. 16, pp. 395–398, December 1987. [Online]. Available: http://doi.acm.org/10.1145/38714.38755

[24] E. Anderson and J. Tucek, "Efficiency matters!" *SIGOPS Oper. Syst. Rev.*, vol. 44, pp. 40–45, March 2010.

[25] VMware, "Performance characterization of vmfs and rdm using a san," www.vmware.com.