Analyzing the Energy Efficiency of a Database Server

Dimitris Tsirogiannis
University of Toronto
Toronto, ON, Canada
dimitris@cs.toronto.edu

Stavros Harizopoulos
HP Labs
Palo Alto, CA, USA
stavros@hp.com

Mehul A. Shah
HP Labs
Palo Alto, CA, USA
mehul.shah@hp.com

ABSTRACT

Rising energy costs in large data centers are driving an agenda for energy-efficient computing. In this paper, we focus on the role of database software in affecting, and, ultimately, improving the energy efficiency of a server. We first characterize the power-use profiles of database operators under different configuration parameters. We find that common database operations can exercise the full dynamic power range of a server, and that the CPU power consumption of different operators, for the same CPU utilization, can differ by as much as 60%. We also find that for these operations CPU power does not vary linearly with CPU utilization.

We then experiment with several classes of database systems and storage managers, varying parameters that span from different query plans to compression algorithms and from physical layout to CPU frequency and operating system scheduling. Contrary to what recent work has suggested, we find that within a single node intended for use in scale-out (shared-nothing) architectures, *the most energy-efficient configuration is typically the highest performing one*. We explain under which circumstances this is not the case, and argue that these circumstances do not warrant a retargeting of database system optimization goals. Further, our results reveal opportunities for cross-node energy optimizations and point out directions for new scale-out architectures.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems – *query processing*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software – *performance evaluation (efficiency and effectiveness)*.

General Terms

Experimentation, Measurement, Performance.

1. INTRODUCTION

In the same way that *performance* has been central to systems evaluation (e.g., measuring completed tasks per unit of time, or Queries per Second), *energy-efficiency* (e.g., completed tasks per unit of energy, or Queries per Joule) is quickly growing in importance for minimizing IT costs. Most recent work in improving energy efficiency in large data centers is either hardware/platform-oriented or workload-management oriented. In the database software field, there have only been a few preliminary studies in this direction [4, 8, 9, 11, 20]. Our aim is to better understand the energy characteristics of database systems on modern hardware.

The focus of this paper is first assessing and then exploring ways to improve the energy efficiency of a single-machine instance of a database server, with standard server-grade hardware components, running a wide spectrum of data management tasks. We focus on the efficiency of a single node in a scale-out (shared-nothing) architecture, a common building block for a variety of data analysis systems, from Hadoop clusters to massively parallel databases.

Our main contributions are as follows:

- A detailed study of the power-performance profiles of core database operators on modern scale-out hardware. To our knowledge, ours is the first study to provide this characterization. Unlike in other contexts [15, 3], we found CPU power does not vary linearly with CPU utilization, and utilization is a poor proxy for CPU power. Surprisingly, we found that the CPU power used by various operators can vary up to 60%, even when they have the same utilization.
- A thorough investigation of the effects of both hardware and software knobs on the energy efficiency of complex queries in two widely used engines: PostgreSQL and System-X[1].
- In almost all cases, we find that unlike what previous studies have suggested, the highest performing configuration is the most energy-efficient. In the few cases where this does not hold, the improvements in energy efficiency are less than 10%. This result is mainly due to the large, up-front power-costs in current server components, as discussed in Section 5.

We purposely do not focus on techniques spanning multiple nodes (e.g., resource consolidation in underutilized clusters [10]) or on alternative energy-efficient hardware configurations (e.g., low-power non-server grade components [1]). These two are promising directions that require extensive treatment, and in the last section of the paper we speculate on how our results can be used towards these explorations.

1.1 Where does power go?

To assess the opportunities for optimization, we start by measuring the power of system components from idle to fully utilized. Figure 1 shows the power break down (measurement details are in

[1]System-X is a popular commercial DBMS.

SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.

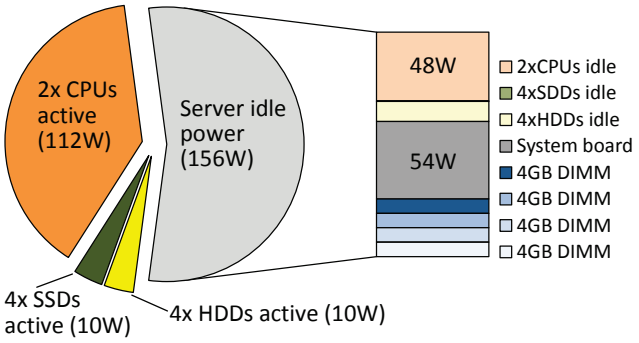

Figure 1: Power breakdown of test machine.

Sections 2 and 3) of one configuration of our 8-core (dual-CPU) test machine. An important factor for any study in energy efficiency is the hardware setup used; Section 2 discusses hardware trends and argues why this particular setup's power profile is close to what we expect to be a typical node in a cluster.

The right half of the pie-chart (grayed out) is the idle power consumption and accounts for about half of the peak power. As the breakdown shows, the three main components contributing to the waste are the idle power of the two CPUs, the fixed power of RAM and the rest of the system-board components. The idle power of the HDDs is relatively small, and the consumption of the SSDs is close to zero. The left half of the pie-chart shows the additional power consumed when all CPUs and disks are fully utilized. The SSDs and HDDs draw similar power, however, the dominant components are the two CPUs (+ 112W). Next, we examine in more detail the components that exhibit measurable dynamic power range, i.e., the difference between idle and peak power.

1.2 Power use and resource utilization

The largest power consumer in our system are the CPUs. These are two quad-core Intel Xeon E5430 CPUs (Harpertown) which use aggressive power management techniques. To our knowledge, there are no studies that show the CPU power use of database-like operators in modern processors as the CPU utilization varies. The literature provides several power models, however, as a recent study shows [15], these models begin to break down in CPUs with shared resources and aggressive power management techniques. Rather than attempting to derive yet another power model, we focus on computations, data access patterns, core and cache sharing patterns, and data sizes that are directly relevant to database operators. To factor out overheads that a complete DBMS may carry, we implement high-performance, micro-benchmark kernels for cache-conscious hash joins and parallel sorting. For scan operators we use an open-source minimum overhead row/column storage manager.

Figure 2 shows the total CPU power consumption for several operators, as we vary the number of cores used, from 1 to 8 (each core is 100% utilized, point 0 is the idle power). In this paper, when we refer to CPU utilization, we mean the fraction of total cores used since each of our micro-benchmark processes will fully utilize a core from the OS perspective. Figure 2(a) corresponds to a performance-oriented process scheduling policy whereas Figure 2(b) uses an energy-conserving policy (details are in Section 3).

Surprisingly, different operators can vary by more than 60% in power consumption (e.g., sort and row-scan using two cores). Section 3 explains this result and presents further details and performance results with these micro-benchmarks. We also find that in

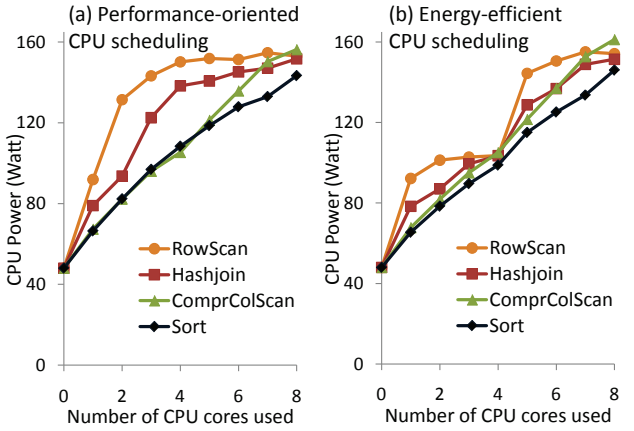

Figure 2: CPU power vs. utilization for database operators.

SSDs, power-use is nearly proportional to utilization (Section 3), something we were not able to find in the literature. These results reveal opportunities for cross-node energy optimizations (e.g., the last 30% of a node's CPU computation capacity comes essentially for free) and inefficiencies in server CPUs that new/alternative hardware should address (e.g., need for smaller CPU caches). Section 5 discusses these and other implications in detail.

1.3 Affecting energy efficiency: what "knobs"?

We define energy efficiency as the ratio of useful work done to the energy used, which is the same as the ratio of performance to power:

$$\text{EE} = \frac{\text{Work done}}{\text{Energy}} = \frac{\text{Work done}}{\text{Power} \times \text{Time}} = \frac{\text{Perf}}{\text{Power}} \qquad (1)$$

As database software is rich in tunable parameters, from system-level constants to query planning and execution, and our experiments so far point to several options that can affect power-use, these parameters can potentially affect energy efficiency. The most promising knobs are the ones that can directly trade CPU cycles for disk access time since these are two resources with significantly different power-use profiles. Such tradeoffs exist in access methods (sequential scans vs. clustered and non-clustered index scans), column-oriented vs. row-oriented record access, compression techniques (lightweight vs. heavyweight), and join algorithms (hash-join or sort-merge vs. nested loop join). Recent work has speculated that some of these knobs may be promising [4, 8], and others have shown specific cases where energy efficiency is improved [11, 9, 20] at the expense of performance.

With all these available knobs at hand, we set to find regions where performance may be sacrificed for energy efficiency. Such tradeoffs would make the case for considering energy apart from pure performance inside a modern, scale-out server node.

Note that, throughout the paper, we focus on operators and queries that commonly appear in data analysis tasks and data warehousing workloads. In online transaction processing (OLTP) workloads, most of the software knobs we consider in this paper are not usually applicable (e.g., long-running scans, joins, different query plans, columnar storage, compression). Although our results do not directly apply, recent research shows [6] that modern OLTP applications can significantly improve transaction throughput, and thus energy efficiency, by redesigning the entire OLTP engine, rather than just tuning it.

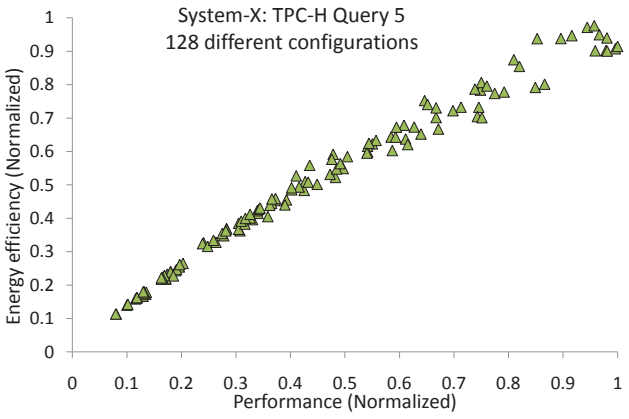

Figure 3: Energy efficiency vs. performance in System-X.

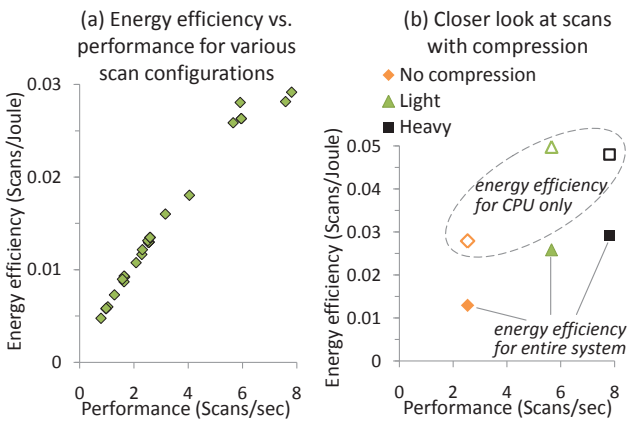

Figure 4: Scan results (including compression).

1.4 Energy efficiency vs. performance

To increase the generality of our results and ensure that we do not miss out on opportunities for improving energy efficiency due to system-specific limitations, we experimented with a total of five systems/database kernels. We implemented two minimum overhead kernels, one for performing parallel, cache-conscious hash joins, and one for sorting, using Alphasort and a custom implementation of parallel merging. We then chose a high-performance, open-source database storage engine that supports both column-oriented and row-oriented database scans along with light-weight compression [7]. Lastly, for full-fledged DBMSs, we chose a popular open-source DBMS (PostgreSQL) and a popular commercial DBMS (System-X).

In the overwhelming majority of our experiments (we collected over 1000 data points) we observe that, for any given database task (from simple scans to multi-user query workloads), *the most energy-efficient configuration is the highest performing one*. We found isolated cases where small gains in energy efficiency can be achieved by choosing a lower performing operating point. Even if we were to assume a zero-power system board (to derive an upper limit) these gains extend to only 10-12%.

Figure 3 shows the energy efficiencies of TPC-H query 5 in a System-X database for 128 different configurations. Note, this graph is a parametric graph in which both axes – energy efficiency (y-axis) and performance (x-axis) — are dependent variables; they both depend on the configuration. We see that energy efficiency goes hand-in-hand with performance. As Section 4 discusses, there is almost always a configuration that can further improve energy efficiency by simply improving performance. Even for the points near peak performance, their energy efficiency varies less than 10%.

Figure 4 shows microbenchmark results that are representative of the picture we see repeatedly in our experiments. The left graph shows the energy efficiency for a set of SSD disk-resident column-store scans. Again, each point corresponds to a different configuration (we varied number of CPUs, CPU frequency, and compression algorithms). As before, the highest performing configuration is the most energy efficient. The right graph isolates three points and shows how the performance and energy efficiency change when employing two different compression methods ("light" stands for FOR and "heavy" for FOR-delta [7]). This is a case where energy efficiency and performance improve at different rates. As a result, if all components other than CPU and disks consumed zero power (depicted as "energy efficiency for CPU only"–we discuss this in Sections 4 and 5), we would have seen a small drop in energy efficiency when switching from a "light" compression method to a faster "heavy" one.

1.5 Analysis of results and implications

Our results come in direct contrast to recent work [8, 9, 20] that has suggested that energy efficiency and performance are often two different optimization goals. Taking a closer look at the results in those papers we can see that the differences stem from computing energy efficiency without taking into consideration the power of peripheral components and the idle power of the CPU. This is a result of the fixed up-front power that most system components exhibit at their lowest performance points, which we explain in Section 5.

Although our results seem to discourage any further software-level energy reduction methods within a single-node DBMS, we believe that they point to opportunities in (a) cross-node energy efficiency techniques for co-locating computation and optimizing data placement and movement, and (b) alternative single-node hardware architectures that remove inefficiencies in current designs.

2. BACKGROUND AND MOTIVATION

Energy efficiency of computing equipment in data centers is an important concern for several reasons. First, powering and cooling costs are starting to overtake the cost of hardware [5]. Second, increased energy use has negative implications for density, reliability, and scalability within a data center. Finally, increased data center energy use has prompted environmental concerns leading governments across the world seeking to regulate enterprise IT power. For these reasons, we are starting to see a shift in industry and research towards optimizing for energy efficiency.

This shift is broad, ranging from chips to data-centers. Chip designers have considered a variety of power-saving techniques such as dynamic frequency and voltage scaling (DVFS), clock routing optimization, asymmetric multi-cores, and so on. System architects have suggested strategies for dynamically managing DRAM power states, disk speed control, or spinning down disks. We are seeing new energy-efficient platform redesigns that meet the performance SLAs of a small, but important class of workloads, e.g., web servers and data analysis [1, 5, 16]. Finally, recent work proposes ensemble-level optimizations, for example, shifting workloads around for meeting power and temperature constraints or consolidation for improved energy use [19]. Going forward, data-center architects are investigating holistic redesigns that treat the data center as a single computer [3, 14].

Component	Min (W)	Max (W)
2xIntel Xeon E5430 Quad Core 2.66GHz	48W	160W
4x4GB FB-DIMMS (RAM)	40W	40W
4x300GB Seagate Savvio 10K.3 2.5"	14W	24W
4x64GB Intel X-25E 2.5" (SSD)	0.2W	10W
System board (remaining components)	54W	54W

Table 1: Configuration of our database server.

Towards this end, some are calling for energy proportional hardware [2]. A perfectly proportional component uses no power when not in use and only uses power in constant proportion to its performance. Since energy efficiency is the ratio of performance to power, energy-proportional hardware provides constant energy efficiency at all performance regimes. With such hardware, we need not employ higher-level techniques to adjust for the most efficient point. The hardware designers provide the best efficiency possible, and the software designers continue to worry about performance.

Today, however, components are hardly energy proportional, but their dynamic power range and proportionality are steadily improving. CPU vendors initially introduced low-power versions of their processors for the mobile market. To improve proportionality for server-class CPUs, vendors started to introduce increasingly finer active and sleep power states. Commercial products already include hooks to control power states of DRAM, and new memory controllers are on the way that dynamically adjust these states. There are also new types of non-volatile memory around the corner like PCRAM [12] and memristor [18], vying to replace DRAM altogether. For drives, we see new drives with multiple spin rates as well as lower power drives with more sleep states. Finally, SSDs are on track to replace hard drives in contexts where capacity is not the limiting factor. Our experiments later will show that these drives exhibit near linear proportionality.

Using these newer components, we still are left with the question of how to best assemble these into larger, energy-efficient systems. There are two main approaches for building data-management architectures: scale-up (shared memory and shared disk) and scale-out (shared-nothing). The former has many connected components: servers and disks, all managed as a single whole. For these, the most energy-efficient approach involves choosing the right balance of components for a task and powering down unneeded resources. Today, however, we are limited to high-end, typically more power-hungry components that are used in such architectures.

On the other hand, scale-out approaches involve picking the best single node configuration and then connecting them via a scalable network. This approach seems more amenable to energy efficiency optimizations for data-centric workloads. We can pick from a variety of components for more aggressive energy optimizations on single node, and then horizontally scale the task to achieve needed performance while maintaining the same energy efficiency. Other projects like FAWN [1] and microslice servers [5] have taken this approach but not for database workloads. Thus, in this paper, we focus on understanding the power-performance tradeoffs for a single database node in a scale-out setting. Others have started to look at such architectures [9, 20], but those analyses fall short. They neither provide a detailed study of database operators on modern CPUs nor do they account for total system power.

Our current single node setup uses modern (as of 2009) CPUs, DRAM, and SSDs. We chose this setup because it is inline with future trends in components: the CPU has several active power states and the SSDs are state-of-the-art. Moreover, the system is balanced for performance; the disks when fully utilized can keep all the cores busy on simple operations like sorting and hashing.

3. WHERE DOES POWER GO?

In this section, we analyze the power profiles of different hardware components in the context of database operations. The goal is to increase our understanding of how these operations affect power consumption and, ultimately, to reveal the energy saving potential. For that purpose, we designed a set of *micro-benchmarks* to exercise the hardware components of a database server using typical database-centric operations such as scans, joins, and sorts.

In this study we used an HP xw8600 workstation running a 64-bit Fedora 4 Linux with kernel 2.6.29. The configuration of our server is presented in Table 1. The system has two Intel Xeon E5430 2.66GHz quad core processors, for a total of 8 cores. Each processor has an 32K L1 instruction and data cache, and two 6MB L2 caches shared by two cores.

In all of our experiments, we used a Brand Electronics 20-1850 CI to measure the total system power. This power meter has $\pm 1.5\%$ accuracy and collects readings once a second. To isolate the power drawn by the SSDs, HDDs, and the CPUs, we attached a clamp meter to the 5V and 12V lines to the components from the power supply. We used an Extech MA120 200A AC/DC mini clamp-on meter which had a resolution of 100mA and an accuracy of $\pm 2.8\%$. By multiplying the current measured with the line voltage, we derive our power measurements. To increase the resolution of the clamp meter, when possible, we wrap the 5V or 12V lines multiple times around the clamp meter's loop, and divide the measurement with the number of loops. Throughout this paper, all reported power and performance measurements are an average over multiple trials.

The last two columns in Table 1 denote the minimum and maximum power consumption of each component. To estimate the total CPU power at load, we run a CPU-intensive computation with a small memory footprint on each core, and attribute the increase in system power from idle entirely to the CPU. Although memory is responsible for 20% of idle power consumption (Figure 1, Section 1), we were not able to measure noticeable variations in power consumption using different workloads that varied the amount of memory accessed and the access patterns applied (sequential vs random memory accesses). Therefore, throughout this study, we assume that the only way to vary memory power use is by physically removing memory modules.

All components other than CPUs, disks, and RAM, are represented as "system board" in Table 1 and collectively consume $54W$ (again, we were not able to measure noticeable differences between minimum and maximum power). While these components include fans, the network card, the GPU, and the motherboard, the majority of the power budget, according to manufacturer specs, is attributed to the memory and IO controller units on the motherboard (we did not exercise the full capabilities of the GPU as we were always remotely connected to the test machine). The total power consumption of our system in idle state was measured at $156W$.

Note that, from a power consumption perspective, we believe our hardware setup closely approximates a node in production-use scale-out architectures. We purposely configured the system with 16GB of RAM and not 32GB, as we only had available DIMMs of 4GB each; these DIMMs consume the same power as the more expensive 8GB ones, which would make it possible to have 32GB of RAM at a $40W$ power budget for main memory. Next-generation chip processes could easily double that amount for the same power budget. Our 8 disks provide an aggregate of 1.5GB/sec bandwidth, and adding four more SSDs could yield an additional 1GB/sec at only a 3.5% increase in total server power.

For each row in Table 1, the difference between maximum and minimum power consumption denotes the dynamic power range (maximum increase in power consumption) attributed to a specific

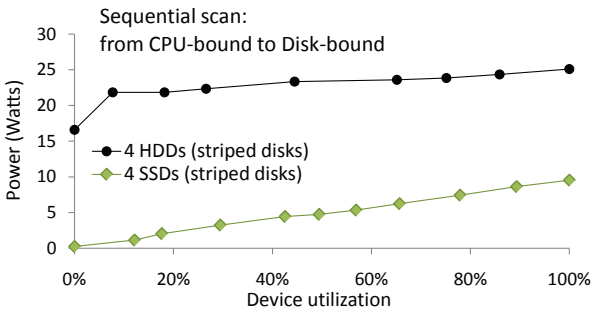

Figure 5: Power consumption of SSDs and HDDs as a function of device utilization.

component. Note however, that dynamic power range does not necessarily reveal opportunities for energy saving, it only indicates the energy saving potential. In general, energy inefficiency stems from systems' poor energy proportionality, i.e., power use does not increase in proportion with performance, which is due to the poor proportionality of individual hardware components. In order to analyze the energy efficiency of database systems, it is important to study first how power consumption varies as a function of each component's utilization. Next, we examine in isolation the two components that exhibit dynamic power range: disks and CPUs.

3.1 Disks

Our database server utilizes both enterprise hard drives (HDDs) and solid state drives (SSDs). Unlike memory, these components exhibit higher power variability, being responsible for 15% of total active power. One common characteristic in both storage technologies is the existence of two operational states, idle and active, each consuming a different amount of power. What is not clear from past work is the energy proportionality (if any) of these devices while executing database-centric operations.

To study the energy proportionality of SSDs and HDDs, we use a minimum overhead row-store/column-store kernel [7] that emulates a row-based or column-based database-storage engine and can also perform simple predicate evaluation and aggregations. The kernel utilizes direct IO to reduce the CPU overhead while accessing data blocks and asynchronous IO to improve device throughput.

To measure the power consumption of SSDs as a function of device utilization, we configured these devices in RAID-0 (striping) and we used our kernel to read a 100GB file sequentially; the block size was set at 128KB. Device utilization was computed as the ratio of measured to maximum throughput. Predicates of increasing complexity were progressively applied to introduce CPU overhead that effectively reduced the throughput measured. Again, we isolated SSD power using the clamp meter. In Figure 5, we plot the power use of the four SSD devices as a function of device utilization. As Figure 5 clearly demonstrates, SSDs exhibit perfect energy proportionality; they consume almost nothing with no load and exhibit linear power increases with additional load.

Figure 5 also shows the power profile of HDDs measured using the same procedure. HDDs are clearly not energy proportional because they consume nearly half the power at idle state and approximately 80% of maximum power when the device becomes active. The initial power cost comes from spinning the disk platter, and we speculate the second jump comes from exercising additional circuitry and caches. Thereafter, HDDs exhibit a linear increase with utilization. As shown in previous work, the poor proportionality of drives can significantly reduce energy efficiency, especially when these devices are under-utilized [11].

3.2 CPUs

Interestingly, the CPUs are responsible for 85% of the power increase at load in our system. Several power models have been proposed that assume a linear correlation between CPU utilization and power consumption [15]. However, linear models for total CPU power are good approximations under very specific conditions: a) operations are cpu-bound, b) there are no shared resources among CPU cores, and c) no power management techniques are applied.

These conditions, however, do not hold for database operations as these are not always CPU-bound. Additionally, in modern multicore CPUs the processing elements (cores) typically share several resources such as part of the cache hierarchy and the memory bus. Furthermore, hardware and software power management techniques are commonly applied. Modern CPUs dynamically vary CPU frequency depending on the workload applied. At the software level, operating systems utilize energy-saving scheduling techniques that minimize CPU power consumption.

Given the aforementioned conditions, it should come as no surprise that linear power models do not accurately predict the power use of modern CPUs running data management tasks [15]. Our goal in this paper is not to develop a new power model for modern CPUs. Instead, we are interested in understanding a) how CPU power is affected by database operations and b) the efficacy of hardware and software power management methods in this context. Towards that goal, we developed a set of micro-benchmarks that perform three classes of database operators, namely hashing, sorting, and scans. The micro-benchmarks are designed to exercise all cores as well as their shared resources, such as CPU cache and memory bus.

3.2.1 Micro-benchmarks

The first micro-benchmark is a custom join kernel that implements a CMP-adaptation of a cache-conscious hash join algorithm [17] for computing the join of two memory-resident relations in parallel. Like the grace hash join, our join kernel proceeds in three steps: a) partitioning, b) build, and c) probe. In the partitioning step, we divide the relations into partitions such that each build partition with its hash index fits into L2 cache (3MB per core). For each partition, we populate a hash index with the build tuples and probe the index with the other relation. Each step is executed in parallel by all cores via horizontal partitioning of the input relations and the intermediate partitions. The build relation holds 100K tuples, and the probe relation holds 20M 100-byte tuples.

The second micro-benchmark consists of a sort kernel that implements two in-memory parallel sorting algorithms tailored to CMP architectures. Both sorting algorithms are cache-conscious variants of AlphaSort [13]. The first variant, called *AlphaSort-S*, produces cache-sized sorted runs in parallel and then merges these runs using a serial merging phase. The second variant, called *AlphaSort-P*, applies a parallel merging phase. The main difference between *AlphaSort-S* and *AlphaSort-P* is that the latter exercises all processing elements throughout the entire execution of the sorting task. The input to both sorts are 120M integers (480MB total).

Our last micro-benchmark for measuring CPU power is the scan kernel used in Section 3.1. In this section, we use it to scan both uncompressed rows in memory and compressed columns on disk. To run these scans in parallel, we simply issue multiple simultaneous scans on different files. The row scan operates over 60M 32-byte rows (1.9 GB total, in-memory), and the column scan runs over 60M 4-byte integers compressed using FOR-delta [7], for a total 64MB on-disk. Each run includes multiple scan iterations (for disk-resident files, direct IO ensures no Linux-cache buffering), to allow our Watt meter to collect several measurements.

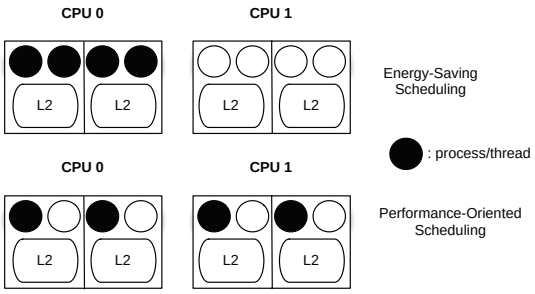

Figure 6: Different process scheduling policies.

3.2.2 *Analyzing Power Consumption*

In this section, we focus mainly on CPU power consumption of the micro-benchmarks running under different scheduling strategies and frequency settings. Section 4 examines energy efficiency in greater detail. To show the effects of process scheduling, we try two different policies: energy-efficient and performance-oriented. The former is a policy in the latest Linux kernel that first co-locates processes on one CPU before using the other to save power. The latter is one we implement by spreading processes across CPUs to maximize memory bandwidth. Figure 6 shows the difference between energy-efficient (top) and performance-oriented (bottom) scheduling on our CPUs. In all figures, the performance-oriented policy is the default, and the energy-efficient policy is labeled.

Figure 2 (Section 1) shows the CPU power for our database operators as we vary the number of cores used. In all cases, each core was fully-utilized when used, and the server automatically adjusted the frequency. Note the sudden power increases when cores 1 and 2 are used in the performance-oriented policy and when cores 1 and 5 are used in the energy-efficient policy. These are the fixed power costs as the CPUs transition from idle states to active. Looking closer at (b), we see that these up-front costs exist for all operators but are much more pronounced for the row scan and hash join.

Although we cannot provide a conclusive explanation for this effect, using Intel's Vtune performance profiling software, we found that operators with those high up-front costs also exhibit higher memory bus utilization. Figure 7 presents the bus utilization of each operator as we vary the number of cores and the scheduling policy. It shows discontinuities at the same points as Figure 2, and the bus is nearly saturated once both CPUs are active for row scan and hash join. We speculate that these operators activate and put more stress on the memory subsystem of the CPU, leading to increased power consumption.

These experiments lead to two important conclusions. First, CPU power is not a linear function of the number of cores used (which is what we mean by CPU utilization). Second, for a fixed configuration (number of cores, scheduling policy), different operators may differ significantly (60% in our experiment) in power consumption. Thus, simple models based purely on utilization are not suitable for predicting the CPU power, let alone system power, when running database operators.

In the next set of experiments, we focus on the power consumption of each individual operator, starting with the join kernel. We experimented with different number of cores, scheduling policies, and CPU frequencies. Our CPU can operate at two frequencies: 2GHz and 2.6GHz, which we manually adjusted and fixed for these experiments. Figure 8 shows the power profiles of the hash join as we varied these parameters. It also shows the same hash join but with a larger build relation of 20M tuples (640 MB), causing each partition to be larger than the L2 cache (non-cache conscious).

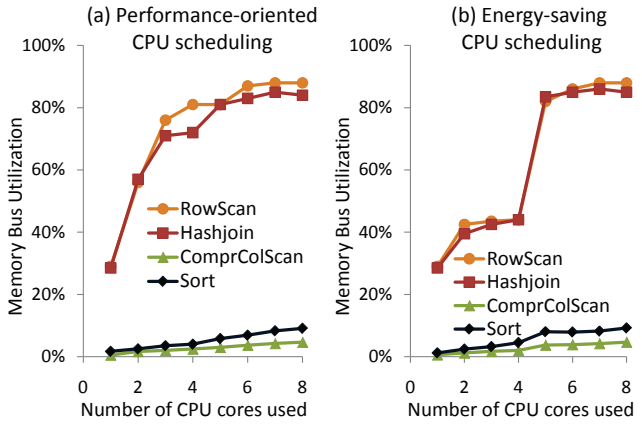

Figure 7: Memory bus utilization of different operators.

This case simulates both a non-cache conscious hash join and an in-memory random index scan. Note, unlike the previous graphs, these are parametric graphs in which both axes depend on the independent variable, number of cores used. The performance (x-axis) is the inverse of the runtime.

There are some interesting effects to note. First, the curves are all concave down (bend down) except when the energy efficient policy activates the second CPU. This is because once one core on one of our CPUs is used, all cores on that CPU transition from idle to active power states. In addition, the CPU activates and exercises other needed shared components like caches and memory interfaces. Thus, we see large power increases when a CPU becomes active, and additional performance comes at little added power cost. Second, lower frequency uses less power but also runs slower, which does not translate into much higher efficiencies as we show later in Section 4. Third, for the non-cache conscious join, the power use is lower for the high frequency curve compared to the cache-conscious join, but the low frequency curves are in a similar power range. This suggests that the cores on the non-cache conscious join are probably stalled on the cache and thus cannot effectively use the additional cycles at high frequencies.

Figure 9 shows our experiments with the sort kernel. We omit the graphs with energy efficient scheduling as it did not make much of a difference. *AlphaSort-P* exhibits an almost-linear increase in power consumption as performance increases. On the other hand, *AlphaSort-S* consumes comparatively less power because when more cores are used, the fraction of time spent on the serial phase increases, and so most of the cores remain underutilized. In both cases, the lower frequency does not make a significant enough difference in power to justify the decreased performance.

Figure 10 shows the same experiments for the scan operators. We see that in-memory row scan has a highly non-linear profile, with several performance points overlapping with each other, unlike any other operator. For clarity we only plot results for the two scheduling policies under high frequency; the low frequency results have a similar shape, shifted towards left and down. This graph now fully explains the power-use profile of in-memory row scan: For performance-oriented CPU scheduling (solid square points, "Hi-freq"), going from one to two parallel scans (1 core to 2 cores in two different CPUs) doubles performance with a large increase in power ($40W$, which is comparable to the $44W$ increase from idle to 1 core). The next two points offer smaller increases in performance and power, as the memory bus to each CPU becomes the bottleneck. With four parallel scans, the memory bus in both CPUs

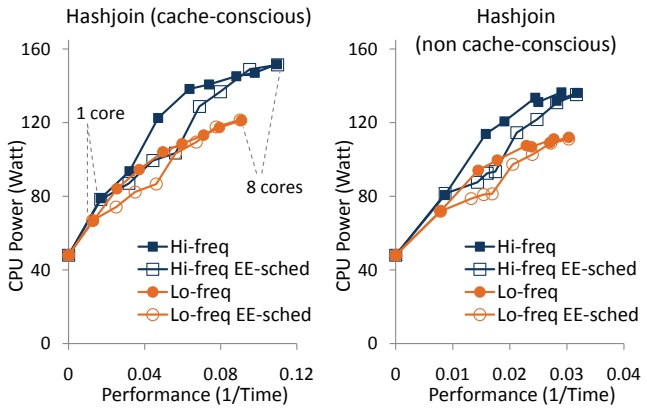

Figure 8: Hashjoin operators.

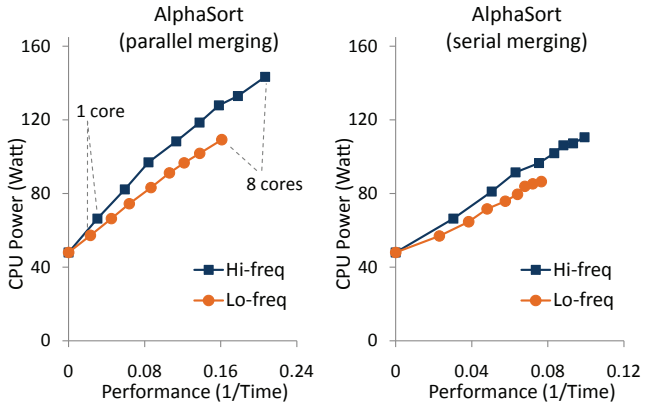

Figure 9: Sort operators.

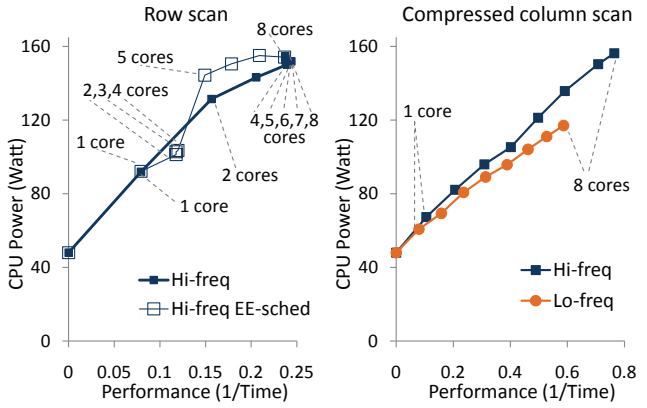

Figure 10: Scan Operators.

saturates and additional scans offer no increases in performance or power (hence the overlap of all solid square points for 4,5,6,7, and 8 cores). With energy-efficient scheduling (empty square points, "Hi-freq EE-sched") the point overlap comes early, for 2,3, and 4 cores, as all these scans share the memory bus of the same CPU. For 5 cores there is again a jump in CPU power as the second CPU (and its memory bus) are activated. Subsequent small increases in performance, when close to peak power, are due to the fact we measure average scan completion time, and some scans finish earlier in the case of EE-scheduling.

Note that hash joins, like row scans, also highly utilize the memory bus (Figure 7), but are able to continue increasing performance with each additional core used, as each core can work on a cache-resident data set. Compressed disk-resident column scans (right part of Figure 10) behave like the sorts, again with the scheduling policy not making much difference. These scans are not bound by the memory bus (they hardly saturate the I/O bus on the four SSDs) as they require more CPU cycles for each byte read (due to both columnar storage and compression). Once again, though, in all scan cases the best performing high-frequency point is the most energy efficient.

4. ENERGY VS PERFORMANCE

In this section, we take a more holistic approach and study how hardware and software "knobs" affect the energy efficiency of database workloads. Towards that goal, we not only revisit our micro-benchmarks from an energy efficiency perspective, but also experiment with a variety of queries on full-fledged database engines: PostgreSQL and commercial System-X.

Although database systems are notorious for the plethora of configurable parameters that they support, in this section, we select those that potentially have the greatest impact on energy efficiency. In particular, we vary the following database-level "knobs": a) algorithm/plan selection, b) intra-operator parallelism (# of cores running a single operator), c) inter-query parallelism (# of independent queries running in parallel), d) physical layout (row vs column scans), e) storage layout (striping), and f) choice of storage medium (HDD vs. SDD). We also vary the platform-level knobs from before: scheduling policies and frequency settings.

4.1 Algorithm/Plan Selection

Recent work has suggested that to find plans that achieve improved energy efficiency, we will need to redesign optimizers to model energy costs and re-target their plan selection criteria [8, 9]. In this section, we test this assumption by measuring the energy efficiency of a wide range of queries that exercise all components of a database server. We use two popular, feature-rich databases: PostgreSQL, an open-source engine, and System-X, a commercial product. These engines implement a wide selection of query evaluation algorithms and utilize a cost-based optimizer. Additionally, they provide interfaces or "hints" for influencing choices during query optimization. After running many complex queries including those from the TPC-H benchmark, we present a sample of the results that summarize our most important findings.

4.1.1 Access Methods

Access methods are the fundamental building blocks of every query plan, and, in many cases, they determine the performance of a query. So naturally, we start by studying how different access methods affect energy efficiency. We measured the performance and energy efficiency of a single (non-parallelized) scan query: *"select * from LINEITEM where L_ORDERKEY* $< value$*"*. Throughout this paper, we ran all queries over TPC-H tables against a 10GB TPC-H database. We ran this query using different selectivities, access methods (index scan, sequential scan) and storage configurations on both systems. We tried six storage configurations, listed in order of increasing sequential bandwidth: *HDD-1, HDD-2, SSD-1, SSD-2, HDD-4, SSD-4*, where the number indicates the number of drives in RAID-0 (striping).

Figure 11 presents the energy efficiency and performance of the scan query running in PostgreSQL. Note, this again is a parametric graph in which the axes depend on the storage configuration and access method. The left and right graphs show the results for se-

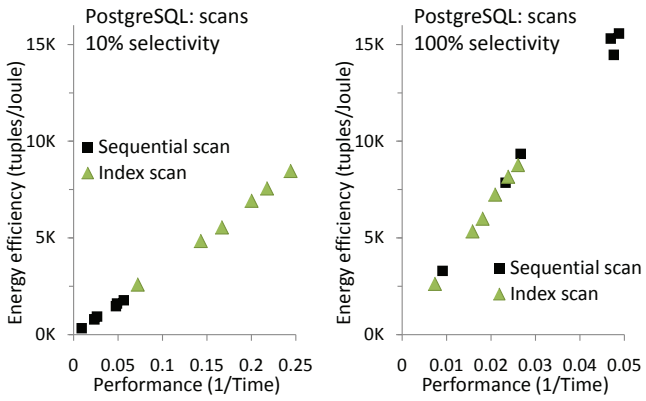

Figure 11: Performance and energy-efficiency of access methods in PostgreSQL.

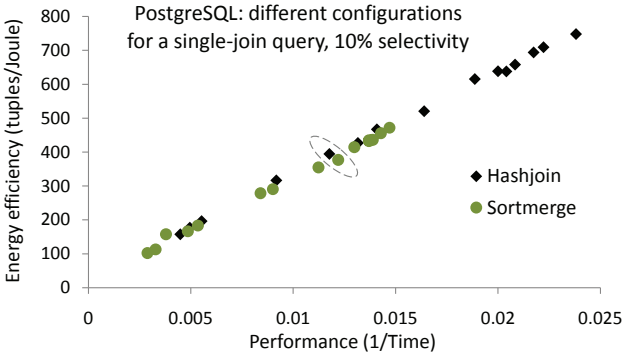

Figure 12: Performance and energy-efficiency of join algorithms in PostgreSQL.

lectivities 10% and 100%, respectively. These graphs clearly show that energy efficiency and performance are linearly correlated. This happens because the dynamic power range among these points is small, only 19% of the minimum $165W$. Thus, power remains relatively constant and energy efficiency varies directly with performance. Interestingly, this graph shows isolated cases that deviate from this relationship. For example, consider the second and third best performing points in the right graph, corresponding to sequential scans on *HDD-4* and *SSD-2*, respectively. *HDD-4* provides marginally better bandwidth, but uses $7W$ more power. The net effect is that the additional power is not worth the bandwidth gains. Nonetheless, the overall efficiency gains are small, less than 6%. We saw similar results for this query in System-X.

4.1.2 Compression

Figure 4(b) from Section 1 shows the energy efficiency of our scan kernel running disk-resident column scans using different levels of compression. Again, we see that when we consider total power, better performance implies better efficiency. If, however, we consider only CPU and disk power, the added power from heavy compression is not worth the performance benefit, though the efficiency gains for the slower algorithm are small.

4.1.3 Join Algorithms

Since joins are one of the central operations in database systems, we next study the efficiency of different join algorithms. To do so, we issued the following (non-parallelized) equi-join query: *"select

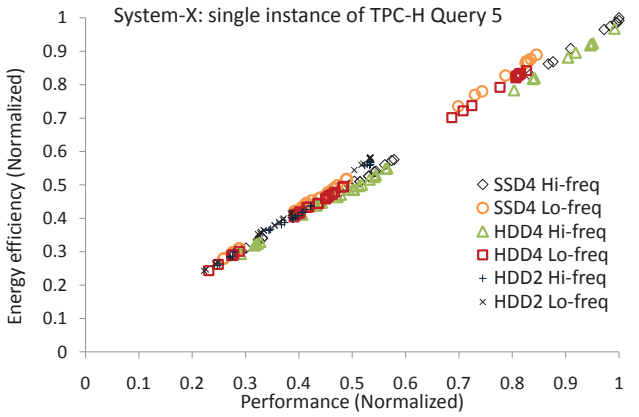

Figure 13: Performance and energy efficiency of different query plans for executing TPC-H Q5 on System-X.

* *from LINEITEM, ORDERS where L_ORDERKEY = O_ORDERKEY and L_ORDERKEY < k"*. We used k to control the join selectivity. We ran this query on both systems using different selectivities, join algorithms (sort merge and hash join), and hardware configurations (CPU frequency, storage system).

Figure 12 shows the energy efficiency and performance of these experiments in PostgreSQL for 10% selectivity. Once again we see a linear relationship because the dynamic power range among these configurations is small, only 14% of the minimum ($169W$) of these. These results are consistent across different selectivities and database engines. Although deviations from the monotonic relationship between performance and energy efficiency exist (e.g., see circle in Figure 12), the difference in efficiency is less than 4%.

4.1.4 Complex Queries and Join Orderings

Given that we saw strong linear relationships for access methods and joins, we hypothesized that more complex queries with different join orders would exhibit the same behavior. To test this assumption, we considered TPC-H Q5 which is a complex five-way join query with sorting and aggregation. We ran this single (non-parallel) query on PostgreSQL and System-X and varied CPU frequency, storage layout (HDD-2, HDD-4, SSD-4), and join orders. Figure 13 shows, as expected, a linear relationship for System X. All points with the same shape are different join orders for a particular hardware setting. Again, the dynamic power range among all these configurations was small, 13% of the minimum $174W$. PostgreSQL showed similar results.

For all non-parallel queries (single process using only one core), variations in CPU power from different single core utilizations combined with variations in storage power do not move the needle compared to the fixed system power costs. These results suggest that in this case, there is no need to change the optimizer.

4.2 Intra-operator Parallelism

With the advent of chip multiprocessors, there has been a continuous effort to leverage on-chip parallelism in order to improve the performance of computationally intensive tasks. In this context, several database operators have been optimized for CMP architectures. Utilizing more cores typically improves the performance of a database operator, but it also increases power consumption. Hence, we next examine the energy efficiency of intra-operator parallelism in the context of CMPs. To do so, we revisit the parallel micro-benchmarks presented in Section 3 from an energy efficiency perspective.

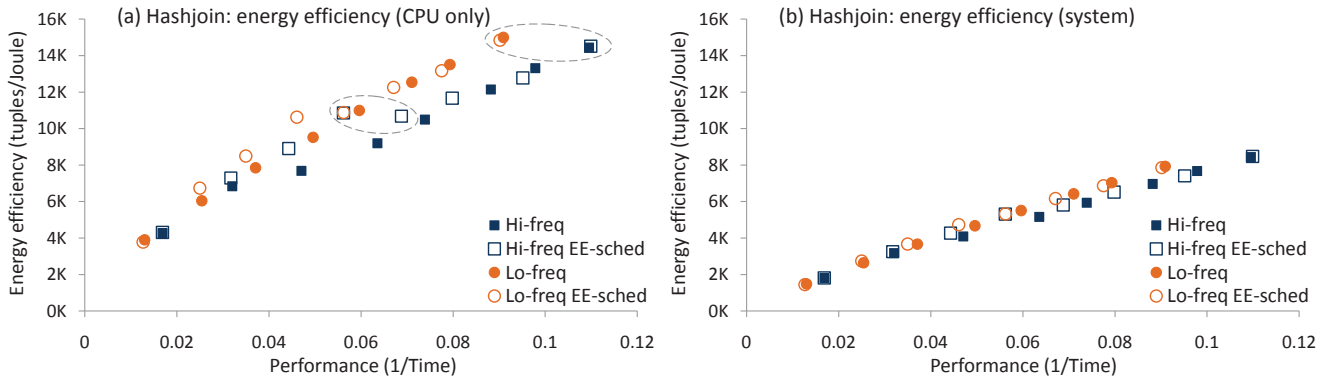

Figure 14: Energy efficiency vs. performance for parallel in-memory hash join, as the number of CPU cores used varies from 1 to 8.

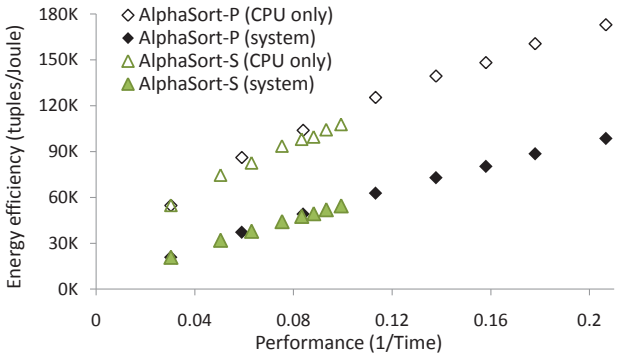

Figure 15: Energy efficiency of parallel in-memory sorting.

4.2.1 *Parallel hash join*

Figure 14 shows the energy efficiency and performance of our cache-conscious hash join as we varied the number of cores used. The left and right graphs compute energy efficiency using only the CPU power and total system power, respectively. The right graph shows a strong linear correlation between energy efficiency and performance, and for points near any given performance level, the efficiency hardly varies.

The left graph shows the energy efficiency curve that only includes CPU power. Again, we see a slightly weaker, but still linear relationship. Also, even though the best performing is not quite the most efficient, the difference is small (see top circle). The other circle shows the points where the energy efficient scheduling moves from using one CPU to two (4 cores to 5). Although performance improves in this case, energy efficiency remains the same because the added power of the other CPU cancels out the performance benefits.

4.2.2 *Parallel Sorts*

In the previous experiment, our join kernel was able to utilize all cores for the entire execution, but this is not always the case. Serial computation and load balancing issues may underutilize CMPs. To understand the energy efficiency implications for these cases, we revisit the parallel sort micro-benchmarks from Section 3. Figure 15 shows the energy efficiency and performance of AlphaSort-S and AlphaSort-P, the serial and parallel merge versions of the sort kernel. Although AlphaSort-S uses less power than AlphaSort-P, it also further underutilizes the CPU as the number of cores increases, which wastes power and leads to much worse energy efficiency. This experiment shows that to reach maximum efficiency,

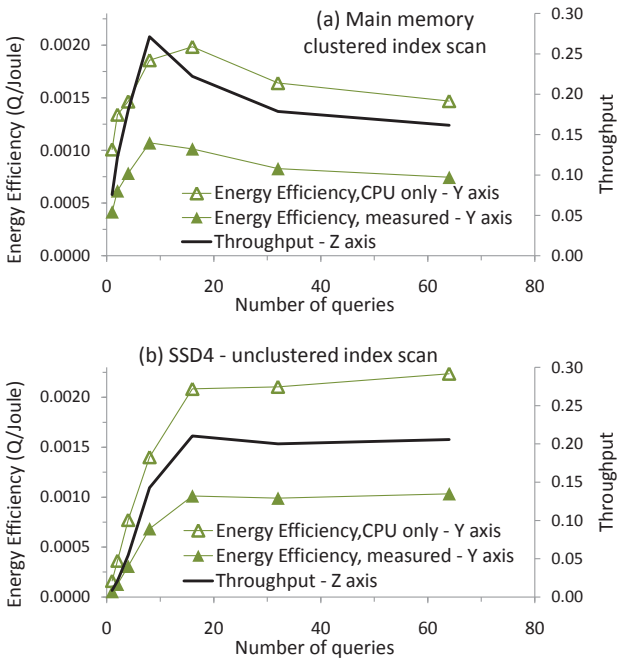

Figure 16: Parallel scan queries in PostgreSQL.

we not only should optimize for performance on a single core, but also optimize for performance by fully utilizing all cores.

Although all these parallel operators show roughly the same energy efficiency-performance behavior as the non-parallel ones, their dynamic power range is large, nearly $92W$. This is 60% of system idle power and 191% of CPU idle power. We initially expected large dynamic range to allow opportunities for trading energy efficiency for performance. Section 5 explains, however, that we see the same behavior because of the concave down nature of the power-performance curves (see Figures 8 and 9) as well as the fixed power costs. For such curves, the relative performance increases are worth the added relative power, so energy efficiency improves with performance.

4.3 Inter-Query Parallelism

In all the previous sections, we studied energy efficiency during the execution of a single query. However, database systems commonly execute multiple queries at the same time. In this section we study how energy efficiency is affected as we increase the con-

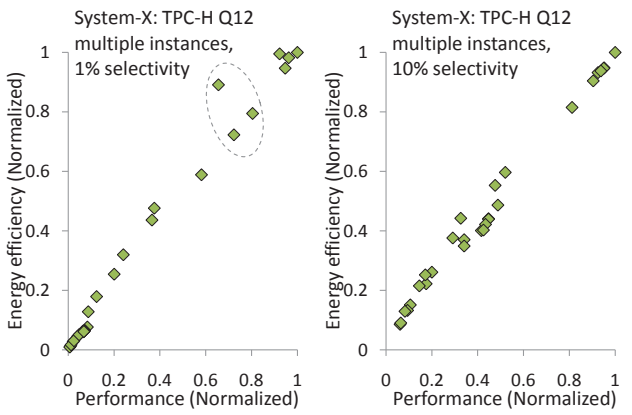

Figure 17: Performance and energy-efficiency of parallel TPC-H *Q12* queries in System-X.

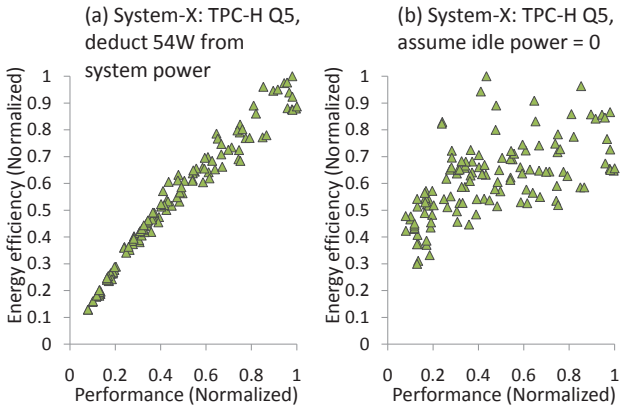

Figure 18: Recalculating energy efficiencies in two hypothetical scenarios.

currency (# of queries running parallel) in the system. Initially, we consider simple scan queries that mainly exercise I/O and the memory sub-system, and then we progressively examine more complex queries that stress all the components of a database server.

We first study the effects of running multiple scan queries concurrently. Figure 4(a) from Section 1 shows the energy efficiency and performance of our compressed column scan kernel from Section 3 on four SSDs. In this experiment, we varied the frequency, compression method, and number of simultaneous scans from 1-8 (i.e., number of cores used). Once again, we see a linear energy efficiency-performance relationship even though the dynamic power range is large.

We next ran many concurrent PostgreSQL scan queries from Section 4.1.1 under various hardware and software configurations. Figure 16 presents the most interesting results. It shows the performance and energy efficiency of a main-memory clustered index scan (top) and an unclustered index scan on 4 SSDs (bottom). In both cases, we see that at a certain concurrency level (8 for clustered, and 16 for unclustered) the throughput peaks and then drops because we reach the capacity of the database software. Once again, energy efficiency, whether including CPU power only or total power, basically follows performance. Note, for the main memory scan, the energy efficiency peaks at slightly higher concurrency when considering CPU power only, but the difference is small.

Finally, to understand the energy efficiency of concurrent complex queries running on our system, we ran two TPC-H queries, *Q12* and *Q5* on System X. The former is a single join query with aggregation, and we ran it under various concurrency levels (1-64), storage layouts, and frequencies. Figure 17 shows the results for 1% and 10% selectivity. Apart from a few deviations, we see the same linear relationship between energy efficiency and performance.

Figure 3 from Section 1 shows the results for Q5 as we varied frequency, CPU scheduling, storage layout (in memory, *SSD4*, *HDD2*, *HDD4*), and number of cores used (1-8). We again see a strong correlation between energy efficiency and performance. Although near peak performance we see some variation in energy efficiencies, the spread is small, less than 10%.

To summarize, regardless of query complexity and what knobs we use, i.e., access methods, operator algorithms, and type and level of parallelism, energy efficiency and performance go hand in hand.

5. ANALYSIS AND IMPLICATIONS

Although Section 3 showed that power-use profiles vary significantly from operator to operator, Sections 1 and 4 showed that regardless of the configuration and execution strategy the most energy-efficient operating points were also the best performing. These results contradict the recent speculation about the need and opportunity for DBMS software to optimize for energy efficiency apart from performance [4, 8], and also contradict recent work that agrees with this speculation [9, 11, 20].

In this section, we explain these disagreements by understanding the conditions under which energy efficiency is optimal and contrast our results with past work in this light. Although our conclusion implies that software-level knob tuning specifically to improve energy efficiency within a single node is not useful, we discuss new challenging single-node problems and promising directions for multi-node techniques.

5.1 The balance of power and performance

Recall from equation 1 that energy efficiency, defined as the ratio of useful work done to the energy used, is equivalent to the ratio of performance to power. As we add system components one at a time, or use more resources, both the power and performance of the system change. Total power increases, and we expect performance to also improve. At any point, energy efficiency improves when the relative improvement of performance is greater than the relative increase of power:

$$\frac{\Delta \text{Perf}}{\text{Perf}} > \frac{\Delta \text{Power}}{\text{Power}} \quad (2)$$

When the two sides are equal, we have a balance of the relative improvements of each, and thus maintain the same efficiency. When the relation reverses, we are in a region of decreasing efficiency — a region of diminishing returns. That is, the increased performance from the additional resource does not justify its power cost.

In our system, there are two main reasons for why the most efficient configuration was typically the best performing. First, as Figure 1 shows, almost 50% of peak power is consumed at idle. This fixed power cost adds a large constant term to the denominator in equations (1) and (2) which makes all subsequent relative power increases worth the added performance, especially when the dynamic power range is small.

Although significant, idle power is not the sole cause. Figure 18(a) recalculates the energy efficiency of System-X from Figure 3, this

time discounting system board power-use. Even in this case, the opportunities for affecting the energy efficiency are small. Figure 18(b) shows the efficiencies using only "active power," that is by subtracting out *all* idle power. Even in this unrealistic case, although the points are more dispersed, the spread in energy efficiency among the best performing points is only 30%. Section 4 also shows similar results when only considering CPU power.

The second reason for this effect is the shape of the power- performance curves for these workloads. Rearranging the terms in the above equation helps explain why:

$$\frac{1}{\text{EE}} = \frac{\text{Power}}{\text{Perf}} > \frac{\Delta\,\text{Power}}{\Delta\,\text{Perf}}. \quad (3)$$

The left hand side (inverse of energy efficiency) is the slope of the line from the origin to the current point (configuration) on a power-performance curve. The right hand side is the "tangent" slope, the slope of the line from the current point to the next point that reflects the added resource. It tells whether the curve bends down, remains steady, or bends up. When the curve is concave down (bends down) or remains steady (linear with non-zero power intercept), the tangent slope is smaller, thus, increasing energy efficiency. But, when the power-performance curve is concave up (bends up), there is an optimal balanced efficiency point after which the tangent slope is larger, thus decreasing energy efficiency.

The power-performance curves in Section 3 of our components are all concave down or steady. The CPU profiles are concave down, especially in the cases of poorly scheduled joins and row scans. This concavity occurs because once the CPUs are active, they activate all cores simultaneously and other resources. As a result, the power "jumps" with each activated CPU and the remaining power increments are smaller. Combining these profiles with proportional SSD profiles still results in concave down profiles. Thus, we find for all our workloads, the most efficient configurations are also the best performing.

Examining two recent efforts that argue for energy efficiency optimizations [9, 20], we see that the analyses are based either on CPU-power only or on the system's "active power." In the former, the margin of improvements is greatly reduced by factoring in the power costs of all components. The effect is even more pronounced in the latter because all fixed costs are omitted.

Lastly, the authors in [11] experimented with a large shared-memory production server running TPC-H. The improvements in energy efficiency came when they removed up to four fifths of their 200+ HDDs. Their power-performance curve as the number of disks varied was concave up. We suspect that as disks were added, average disk utilization dropped leading to only marginal improvements in performance. Meanwhile, each disk contributed a fixed power cost, and after a point, the added performance was not worth it. In contrast, all fixed power costs were upfront in our system, not as additional resources were used. We speculate that if that shared memory system had used SSDs, the energy proportionality of SSDs would have enabled them to avoid these inefficiencies, and it too would exhibit maximum energy efficiency at best performance.

5.2 Implications for database computing

5.2.1 One (hardware) size fits all?

Although the notion of "one-size-does-not-fit-all" in database software design is steadily gaining acceptance, it comes as a surprise that server hardware designs show a striking adherence to the one-size-fits-all philosophy. The last significant change in data-management server architecture was the shift to shared-nothing clusters, largely motivated by price-performance considerations. For modern internet-scale applications, scaling out by orders of magnitude is significantly cheaper than scaling up a large SMP server.

Energy costs, however, have the potential to be the catalyst towards yet another turn in hardware design of large-scale platforms. Recent work [1, 16] has demonstrated that, for random retrieval and data analysis application, significantly different node architectures would be better from a power-performance perspective. If indeed we are heading towards a collection of heterogeneous nodes in a data center, each cluster optimized for specific classes of applications, then what characteristics should these nodes have for database applications? We believe this is a promising direction for future work. Based on the results from Section 3 we can add the following to the list of desired characteristics:

(a) CPUs: high-parallelism, low-frequency, small caches, simpler designs. Database systems are already well-equipped to handle available hardware parallelism. Since CPUs pay a high start-up power cost, after which, each additional active core comes at smaller relative increases in power, database-optimized architectures should leverage high levels of parallelism within and across CPUs. Although in our experiments lowering the CPU frequency did not improve energy efficiency, it did not worsen it either. This is important, because a lower operating frequency can reduce the fixed CPU power cost, which may allow for more energy trade-offs.Lastly, database operators do not always need the CPU cache (or they can be made to work with smaller caches). Therefore, simpler CPU designs with fewer levels in the memory hierarchy may lead to more efficient designs for database workloads.

(b) Solid state storage. SSDs (and we expect any other upcoming electronic non-volatile storage technology) are already close to ideal energy proportionality, as we saw in Section 3. Therefore any energy-efficient node architecture should be based on SSDs and not on mechanical HDDs which necessarily carry a fixed power cost (due to disk spinning). Since we expect HDDs to remain for at least a few years the cheapest option for raw storage capacity, it will be important to focus on techniques that reduce storage requirements to lower the initial investment in solid state storage. Example of such techniques are: compression (even lossy), de-duplication, giving up meta-data storage structures, and storing data summaries instead of exact data.

5.2.2 Shared-nothing, everything, or in-between?

Since we have shown limited opportunity for software-based energy optimizations within a node, we look to the multi-node case. There are two traditional designs to consider: shared nothing and shared disk. A promising research direction is to consider energy optimizations across an entire cluster. Clusters provide extreme scalability. They are, however, often over-provisioned for most tasks, and resources go unused wasting power [3]. Unfortunately, data availability is important, and in a shared nothing environment storage and computing power are coupled. We must find a way to turn off nodes and still maintain data access when the cluster is underutilized. Physical design, replication, hardware-level mechanisms, and data movement are all ripe for investigation. Although current ongoing work in this area considers physical design [10], little work considers dynamic data movement. Databases can easily reassign work among nodes, but the cost of moving data around may exceed the gain in energy by suspending nodes.

An alternative is to use a shared disk design, which decouples storage from compute. If equipped with SSDs, the storage system becomes more energy proportional, and what remains is affecting compute energy proportionality through consolidation [19]. Our experiments show that operators like hash join can force a CPU to consume power disproportionately to its utilization, so consolida-

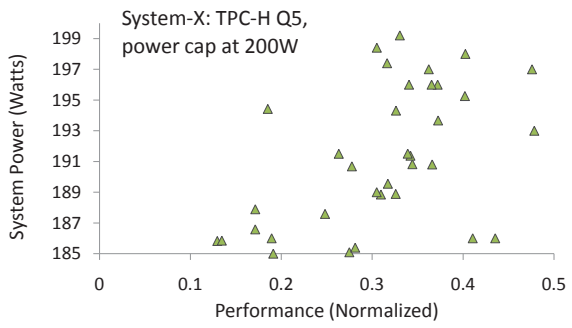

Figure 19: Performance of different configurations of Q5 when power target is set below 200W.

tion would help to avoid waste power. To enable this, we would need accurate models for predicting power consumption. Finally, since shared disk is limited in scalability but can offer simplified energy management, a hybrid approach with clusters of shared disk systems might prove to be the most practical.

5.2.3 Controlling peak power

Another important consideration in data centers is peak power consumption. Racks, especially older ones, have limited capacity to deliver power and, when overdrawn, can cause fuses to trip. Peak power use can also increase temperature beyond the capacity to cool or significantly increase the cost of the cooling. As a result, data centers often need to enforce power budgets and rely on coordinated controllers at all levels from the rack to the node to enforce them [14]. At the node level, hardware mechanisms can regulate power quickly, but have little insight into application performance.

A challenging problem is for database software to tune execution to work within a target power envelope while still maximizing performance. Figure 19 re-plots the points from the various configurations for Q5 on System-X, this time by showing only configurations that consume power just below 200W. Although any of those configurations lies within the target for power consumption, performance can vary up to 4x. Using software mechanisms to cap power consumption while maximizing performance shows promising potential for future research.

6. CONCLUSIONS

In this paper, we study the power-performance characteristics of various database analytic workloads on a modern server intended for scale-out architectures. We are first to detail the power profile of core database operations like scans, hash joins, and sorts. Our results show that the CPU power used by different operators can vary widely, by up 60% for the same CPU utilization, and that CPU power is not linear with utilization. We also experiment with two widely deployed database systems, PostgreSQL and a popular commercial DBMS, varying both software and hardware knobs to understand their energy efficiency effects. In most of our experiments, we found that the best performing configuration was also the most energy efficient. In the few cases where this did not hold, energy efficiency did not improve by more than 10%. This relationship is a result of large up-front power costs in modern server components. Our investigations, however, do point to two promising directions for energy efficiency improvements: resource consolidation across underutilized nodes to save power without sacrificing performance, and alternative energy-efficient hardware to lower fixed-power costs.

Acknowledgements

We would like to thank the SIGMOD reviewers for their comments and the following people for their feedback and helpful discussions: Alexandros Batsakis, Jichuan Chang, Nick Koudas, Christos Kozyrakis, and Partha Ranganathan.

7. REFERENCES

[1] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. Fawn: a fast array of wimpy nodes. In *SOSP*, pages 1–14, 2009.

[2] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *IEEE Computer*, 40(12):33–37, 2007.

[3] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ISCA*, 2007.

[4] G. Graefe. Database servers tailored to improve energy efficiency. In *Software Engineering for Tailor-made Data Management*, pages 24–28, 2008.

[5] J. Hamilton. Internet-scale data center power efficiency. In *CIDR*, 2009.

[6] S. Harizopoulos, D. J. Abadi, S. Madden, and M. Stonebraker. Oltp through the looking glass, and what we found there. In *SIGMOD*, 2008.

[7] S. Harizopoulos, V. Liang, D. J. Abadi, and S. Madden. Performance tradeoffs in read-optimized databases. In *VLDB*, pages 487–498, 2006.

[8] S. Harizopoulos, M. A. Shah, J. Meza, and P. Ranganathan. Energy efficiency: The new holy grail of data management systems research. In *CIDR*, 2009.

[9] W. Lang and J. M. Patel. Towards eco-friendly database management systems. In *CIDR*, 2009.

[10] J. Leverich and C. Kozyrakis. On the energy (in)efficiency of hadoop clusters. In *HotPower*, 2009.

[11] J. Meza, M. A. Shah, P. Ranganathan, M. Fitzner, and J. Veazey. Tracking the power in an enterprise decision support system. In *ISLPED '09*, pages 261–266, 2009.

[12] Numonyx. Phase change memory (pcm): A new memory technology to enable new memory usage models. Online, 2009. http://www.numonyx.com/ Documents/WhitePapers/ Numonyx_PhaseChangeMemory_WhitePaper.pdf.

[13] C. Nyberg, T. Barclay, Z. Cvetanovic, J. Gray, and D. Lomet. Alphasort: a cache-sensitive parallel external sort. *The VLDB Journal*, 4(4):603–628, 1995.

[14] R. Raghavendra, P. Ranganathan, et al. No power struggles: A unified multi-level power management architecture for the data center. In *ASPLOS*, 2008.

[15] S. Rivoire, P. Ranganathan, and C. Kozyrakis. A comparison of high-level full-system power models. In *HotPower*, 2008.

[16] S. Rivoire, M. A. Shah, P. Ranganathan, and C. Kozyrakis. Joulesort: a balanced energy-efficiency benchmark. In *SIGMOD '07*, pages 365–376, 2007.

[17] A. Shatdal, C. Kant, and J. F. Naughton. Cache conscious algorithms for relational query processing. In *VLDB '94*, pages 510–521, 1994.

[18] D. Strukov, G. Snider, D. Stewart, and R. S. Williams. The missing memristor found. *Nature*, 453:80–83, 2008.

[19] N. Tolia, Z. Wang, et al. Delivering energy proportionality with non energy-proportional systems – optimizing the ensemble. In *HotPower*, 2008.

[20] Z. Xu, Y. Tu, and X. Wang. Exploring power-performance tradeoffs in database systems. In *ICDE*, 2010.