# Memory Energy Management for an Enterprise Decision Support System

Karthik Kumar
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907
kumar25@purdue.edu

Kshitij Doshi
Software Services Group
Intel Corporation
Chandler, AZ 85226
kshitij.a.doshi@intel.com

Martin Dimitrov
Software Services Group
Intel Corporation
Chandler, AZ 85226
martin.p.dimitrov@intel.com

Yung-Hsiang Lu
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907
yunglu@purdue.edu

***Abstract*—Energy efficiency is an important factor in designing and configuring enterprise servers. In these servers, memory may consume 40% of the total system power. Different memory configurations (sizes, numbers of ranks, speeds, etc.) can have significant impacts on the performance and energy consumption of enterprise workloads. Many of these workloads, such as decision support systems (DSS), require large amounts of memory. This paper investigates the potential to save energy by making the memory configuration adaptive to workload behavior. We present a case study on how memory configurations affect energy consumption and performance for running DSS. We measure the energy consumption and performance of a commercial enterprise server, and develop a model to describe the conditions when energy can be saved with acceptable performance degradation. Using this model, we identify opportunities to save energy in future enterprise servers.**

## I. Introduction

Energy efficiency of enterprise servers has become a major concern in recent years due to increasing power and cooling costs, and environmental impacts of data centers. These servers must meet strict power, cooling, and thermal constraints. It has become increasingly important to identify power-hungry components in a server, and apply power management techniques to these components. Figure 1 shows the power breakdown in several servers. The processors and the memory are major power consumers; in IBM's Power7 server, memory consumes 46% power, more than the processors' 41%. From Power6 to Power7, memory power consumption increases (as a fraction of the total power), but CPU power consumption decreases. It has been projected that for future computing systems, with more sockets and more memory per core, the memory subsystem will be the single largest source of power dissipation [1].

Power management for memory is less developed compared with CPU power management. Existing low-power modes in memory, such as power-down, still consume large amounts of power. Deeper low-power modes like self refresh are hardly visited due to long exit latencies [2]. It is more difficult applying conventional power management techniques (slowing down and shutting down idle or unused components) in memory for the following reasons: (1) Memory references have both temporal and spatial variations. If we want to shut down a part of memory, we need to predict both *when* and *where* memory may be referred to by an application program. (2) When the CPU is slowed down by a certain factor, it is easier to predict the impact of the slowdown on the application's performance. As a first-order approximation, if a processor reduces its frequency by half, the execution time doubles. Meanwhile, the power consumption is reduced to one eighth. This predictability —how much the application may slow down— is one main reason why dynamic voltage and frequency scaling for CPUs is widely accepted. In contrast, power management in memory does not have such predictable properties and it is difficult to generalize performance and energy of workloads with different memory configurations.

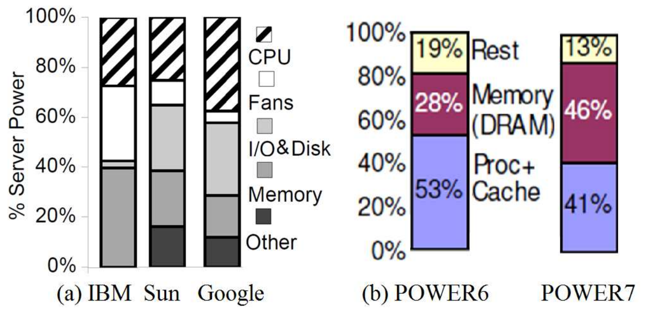


Fig. 1. (a) Power consumption breakdown for different servers from IBM, Sun, and Google (reproduced from [3]). Memory consumes significant amounts of power. (b) Increase in memory power consumption is observed from IBM'S Power6 to Power7 architectures (reproduced from [4]). Memory is the largest power consumer in IBM's Power7, exceeding CPU power consumption.

A program's *memory footprint* refers to the amount of memory referenced during the program's execution. Enterprise applications usually use large amounts of memory, and have time- and space- varying footprints. In the past, performance was the most, probably the only, important consideration for enterprise applications. Hence enterprise servers have as much memory as possible, subject to only budget and capacity constraints. The memory is configured with the minimum latency and the highest bandwidth; this is achieved by powering all available memory and is called the *peak configuration* in this paper. However, in many run-time scenarios, the performance degradation may be negligible when less memory or slower memory is used. This is because enterprise applications, such as databases, have highly advanced memory management, and the ability to modify their query execution and caching strategies based on the amount of memory available. Hence there are scenarios where the application may provide energy-proportional performance at more power-efficient memory configurations than the peak configuration. However, few studies are available examining the power-performance tradeoffs for different memory configurations on a commercial server.

This paper presents a case study of the relationships between memory, performance, and energy consumption of a server running a decision support system (DSS). We use a Nehalem-EP 2-socket server (3 channels in each socket) with 16 Intel Xeon 5500 processors, 48 GB of DDR3 SDRAM memory, and Solid State Drives (SSDs) for storage. The 48GB of memory is organized in 12 DIMMS of 4GB each. Each DIMM contains 2 ranks of memory. Table I shows details about the memory and cache used. We use PostgreSQL [5], an open-source database system, and the DBT-3 benchmark [6] with industry standard DSS queries. We use hardware sampling tools to obtain different memory references and read Model-Specific Registers (MSRs) to obtain information about the memory configuration. In this study, we vary the memory latency and capacity, to identify opportunities for energy reduction with acceptable performance impacts. Using the measurement results, we develop a model to describe the relationships among these factors. Our model suggests that it is possible to save 10-25% system energy by using adaptive memory configurations. We also analyze the performance impacts of adapting the memory configuration, such as reductions in bandwidth and different mappings of physical addresses. We believe this is the first study to quantitatively examine the power-performance tradeoffs of a commercial enterprise application using different memory configurations.

## II. Related Work and Contributions

### A. Background

Memory is spatially organized into channels, ranks, banks, rows and columns as shown in Figure 2. Each channel has a memory controller, and the channels are populated with ranks, typically of 2GB each. The rank is the smallest unit of power management. This implies that if memory is to be transitioned into a low power state, it can only be done at the rank level. Within the rank, there are banks, rows, and columns.

When there is a last level cache miss (Figure 2), the physical address is used to reference a specific channel, rank, bank, row, and column in memory. A permutation of bits of the physical address is used to determine *which bits* index the channel, rank, bank, etc. Typically, the lower order bits of the physical address are used to index the channel and rank. Successive (temporally adjacent) memory references are more likely to have variation in the lower order bits than the higher order bits. Using the lower order bits to index channel and rank results in more parallelism in memory references. This is because each channel has a memory controller, and temporally adjacent memory references are sent to different memory controllers and different ranks. However, this parallelism also means that a large amount of memory may be "touched" to reference a single page in virtual memory.

Many studies have been conducted for dynamic voltage and frequency scaling; however most studies focus on processors. As mentioned earlier, memory power management is more challenging. Bi et al. [7] propose delay hiding in energy management for DRAM. They predict which ranks can be transitioned into low power states for enterprise workloads. Their approach is orthogonal to ours because we reduce the memory footprint (and hence the total number of ranks that are touched) of the application. Qureshi et al. [8] propose phase change memories, an alternate memory technology to alleviate DRAM power consumption. A study from Intel [9] shows that most non-virtualized workloads operate at less than peak memory capacity, and examines how this can save energy for virtualization. Our paper motivates the need to adaptively vary the capacity and frequency of memory for enterprise workloads. Prior work has also addressed application level variations; for example, Dhodapkar et al. [10] compare different techniques to detect program phases.

Prior work has addressed making virtual memory energy efficient. Lebeck et al. [11] propose power aware page allocation; Huang et al. [12] propose power aware virtual memory. Both highlight the importance of OS-level memory mapping. They suggest mapping frequently used memory pages onto "hot" ranks so that other ranks may remain in low power modes. In practice, this is not adopted because most applications and operating systems do not support energy-efficient memory mapping. However the principle of involving the application in memory usage is still important. Our case study is based on this principle, and goes beyond previous works by quantifying the potential energy savings, and the power-performance tradeoffs for adaptive memory provisioning using a DSS application.

### B. Contributions

This paper has the following contributions: (1) We quantify the power and performance implications of applying memory frequency scaling, and reducing the memory footprint of a DSS application on an enterprise server. Previous studies use simulation but we use physical measurements of performance and power in our analysis. We also quantify the effects when giving the application a smaller amount of memory. The purpose of this paper goes beyond identifying that workloads have drastically different reference patterns; rather, we use our experiments to provide evidence for rethinking memory design. (2) We provide a mathematical model to relate system level energy efficiency with application performance and memory power consumption. Our model quantifies how much overhead is allowable, in order to transition memory to a low-power configuration and still be energy efficient. (3) We make the case for an energy-efficient memory design that is adaptive to spatial and temporal variations in workload behavior. Using hardware sampling of memory references, we provide evidence that supports this claim.

## III. Power-Performance Tradeoffs for Memory Configurations

### A. Memory Configurations

The system setup has been explained in Section I. We vary the memory configurations by changing the capacity and the frequency of operation. Table II shows the different configurations considered in this study. $M_p$ is the peak configuration. There are three different capacity configurations ($M_{c1}$, $M_{c2}$, $M_{c3}$) with lower capacities and two different frequency

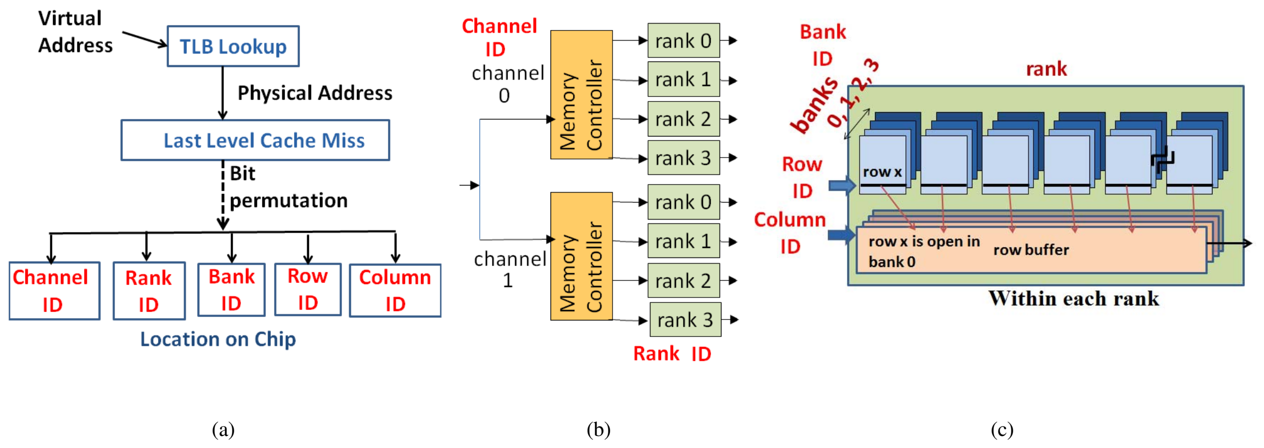


Fig. 2. Spatial organization of memory into channels, ranks, banks, rows and columns. (a) When there is a last level cache miss, different bits from the physical address are used to index the channel, rank, bank, row and column in memory. (b) Each channel has a memory controller, and contains DIMMs. A dual ranked DIMM has 2 ranks. (c) Within the rank, the bank, row, and column ids are used to reference memory.

| Memory Organization | | Cache Organization | |
|---|---|---|---|
| Sockets | 2 | L1 Cache size | 32KB |
| Channels/Socket | 3 | L2 Cache size | 256KB |
| Ranks/Channel | 4 | L3 Cache size | 8MB |
| Banks/Rank | 8 | L1 Associativity | 8 way |
| Rows/Bank | 16384 | L2 Associativity | 8 way |
| Columns/Row | 2048 | L3 Associativity | 16 way |

TABLE I
MEMORY AND CACHE ORGANIZATION OF THE SERVER USED IN THIS STUDY.

| | Capacity | Frequency |
|---|---|---|
| $M_p$ | 48GB | 1066MHz |
| $M_{c1}$ | 36GB | 1066MHz |
| $M_{c2}$ | 24GB | 1066MHz |
| $M_{c3}$ | 12GB | 1066MHz |
| $M_{f1}$ | 48GB | 866MHz |
| $M_{cf2}$ | 24GB | 866MHz |

TABLE II
MEMORY CONFIGURATIONS USED IN THIS STUDY.

| Symbol | Definition |
|---|---|
| $P_{S_p}$ | total system power of $M_p$ |
| $P_{S_\alpha}$ | total system power of $M_\alpha$ |
| $P_{M_p}$ | memory power of $M_p$ |
| $P_{M_\alpha}$ | memory power of $M_\alpha$ |
| $P_\triangle$ | power to transition from $M_p$ to $M_\alpha$ |
| $t_\triangle$ | time to transition from $M_p$ to $M_\alpha$ |
| $t_p$ | application execution time with $M_p$ |
| $t_\alpha$ | application execution time with $M_\alpha$ |

TABLE III
SYMBOLS AND DEFINITIONS.

configurations ($M_{f1}$, $M_{cf2}$) at a lower frequency. The low capacity configurations are obtained by restricting the amount of memory that is available to applications. This may be done by using the "mem" parameter in the Linux kernel, or by depopulating the memory; we found no performance difference between the two methods. The low frequency configuration is obtained by throttling down Model-Specific Registers (MSRs).

We install PostgreSQL, a leading open source database system [5]. For the Decision Support System, we use the DBT-3 workload kit from the Open Source Database Labs (OSDL) [6]. This workload is representative of the TPC-H workload [13], a benchmark used to model database queries for business or organizational decision-making activities. The database files occupy around 80GB on the SSDs. We instrument the motherboard to measure the power consumed by individual DIMMs, and we measure the application performance. The observations obtained from our measurements is presented in the next section.

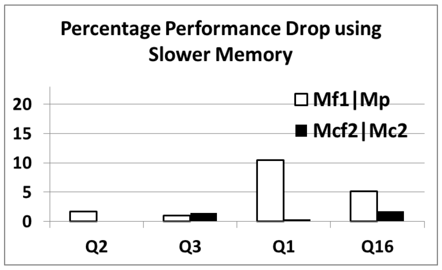
## B. Experimental Data

In this section, we present our measurements of memory power and application-level performance. The DBT-3 workload has 22 different sets of queries: we ran the queries under the different memory configurations. Then, we picked three sets of queries that appeared to be the most sensitive, least sensitive, and moderate, to the memory configuration. Section III-B1 describes the observations for the low frequency configurations $M_{f1}$ and $M_{cf2}$, and Section III-B2 describes the low capacity configurations $M_{c1}$, $M_{c2}$, and $M_{c3}$.

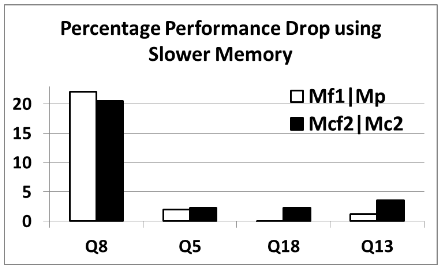
*1) Varying Memory Frequency:* Figures 3 (a), (b), and (c) show the percentage performance degradation of $M_{f1}$ from $M_p$ and $M_{cf2}$ from $M_{c2}$. Some queries are not sensitive to memory latencies. For example, Q2, Q3, Q5, Q18, Q13, Q17, and Q22 have negligible performance degradations. Q16, Q9 and Q20 have performance degradations of less than 5%, Q1 has a performance degradation of 10% at $M_{f1}$, but no performance degradation at $M_{cf2}$. Q8 has a performance degradation of $\approx$ 20% at both $M_{f1}$ and $M_{cf2}$.

*2) Varying Memory Capacity:* Figures 3 (d), (e), and (f) show the percentage performance degradation of $M_{c1}$, $M_{c2}$, and $M_{c3}$, all from $M_p$. There is no visible correlation between the queries that are capacity sensitive, and the queries that are frequency sensitive. The queries in (d) and (e) have low to moderate performance degradations at reduced capacities, and the queries in (f) have large performance degradation at the reduced capacity. We also observed that the database system changed the query plan for some queries when the capacity was reduced. For example, Q2 used hash joins at higher capacities, and nested loops at lower capacities.

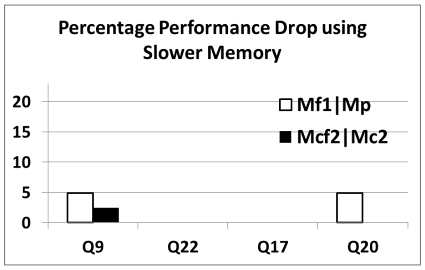
*3) Memory Power Measurements:* We measure the power consumption of the DIMMs; Figure 4 shows the memory power consumption of $M_p$ and $M_{f1}$. We find that an active DIMM (4GB of memory) consumes approximately 4.6 Watts of power at $M_p$. The average power is $\approx$ 10% less at $M_{f1}$,



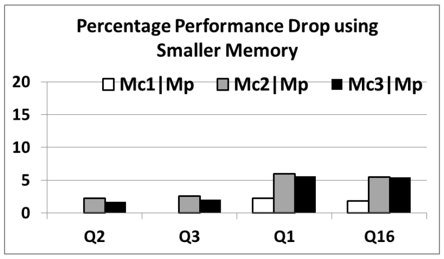
(a)



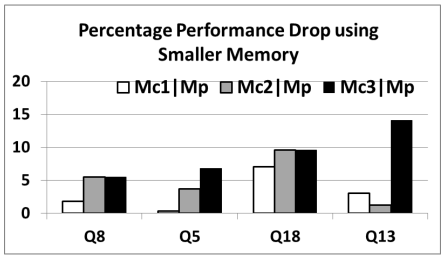
(b)



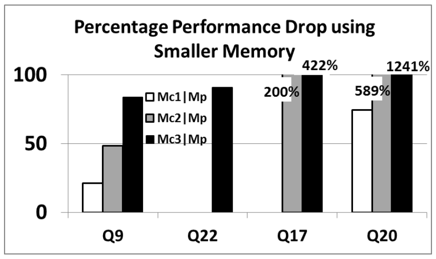
(c)



(d)



(e)



(f)

Fig. 3. DBT-3 Queries performance degradations. A|B means the percentage performance degradation of A with respect to B. (a), (b), and (c) show the performance degradation using slower memory. (d), (e), and (f) show the percentage performance degradation using smaller memory. (f) has a different scale on the yaxis (0-100%).

with the average power per DIMM being $4.14$ Watts. The total power consumed by memory depends on the amount of memory that is powered, and thus the total power of the other configurations can be obtained by linear scaling.

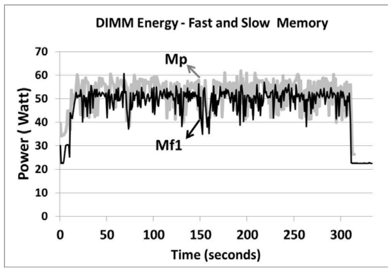


Fig. 4. Rank power consumption for $M_p$ and $M_{f1}$.

## C. Quantitative Analysis

Sections III-B1 and III-B2 show the performance degradation at different memory configurations; Section III-B3 shows the power consumption. This section presents a model describing the conditions when it is energy-efficient, at the system level, to adaptively change the memory configurations. Table III shows some of the symbols used and their definitions. The objective is to reduce system-level energy with negligible performance degradation. The energy consumption running an application for $t$ seconds is $E_S = P_S \times t$, where $P_S$ is the total power. $P_S$ may be expressed as the sum of three components: (1) $P_C$, the average CPU power, (2) $P_M$, the average memory power, and (3) $P_O$, the average power consumed by the remaining components including solid state drives, fans, etc. Thus we have

$$E_S = (P_C + P_M + P_O) \times t \tag{1}$$

The total system power can vary greatly depending on how the CPUs and memory are utilized. Since we only alter the memory configuration, we make an approximation that the remaining system power is a constant. For this analysis, we assume that the memory power is 40% of the total system power, as suggested in [3], [4], [2]. Note that this fraction could be greater or lower, depending on how the CPUs are utilized by the application, and how aggressively unused CPUs are moved into deep low power states.

Now we let the peak configuration of memory be $M_p$, and a lower power configuration be $M_\alpha$, ($\alpha$ can be $c1$, $c2$, $c3$, $f1$, or $cf2$). We need to examine when it is energy efficient to transition the memory from $M_p$ to $M_\alpha$. The execution times of the application using the two memory configurations are $t_p$ and $t_\alpha$. There may be an overhead in transitioning from $M_p$ to $M_\alpha$; we use $P_\Delta$ and $t_\Delta$ to denote the system level power and time for the transition overhead. The transition from $M_p$ to $M_\alpha$ is energy efficient if the following inequality holds

$$\gamma_E = \frac{P_{S_p} \times t_p}{P_{S_\alpha} \times t_\alpha + P_\Delta \times t_\Delta} > 1 \tag{2}$$

Since we assume that memory consumes 40% of the total system power ($P_{M_p}$=$0.4 \times P_{S_p}$), the remaining system power is written as $0.6 \times P_{S_p}$. It is the same in the denominator since we change only the memory configuration. Thus

$$\frac{(P_{M_p} + 0.6 \times P_{S_p}) \times t_p}{(P_{M_\alpha} + 0.6 \times P_{S_p}) \times t_\alpha + P_\Delta \times t_\Delta} > 1 \tag{3}$$

The ratio
$$\gamma_T = \frac{t_p}{t_\alpha + t_\Delta} \tag{4}$$
gives the performance degradation due to the transition and typically we want this ratio to be bounded.

*1) Reducing Memory Frequency ($\alpha$=$f1$):* The memory controller can change the operating frequency by modifying MSRs with very small overhead, and thus we assume that $t_\Delta \approx 0$ in equations (3) and (4). Section III-B3 showed that $P_{M_{f1}} \approx 0.9 \times P_{M_p}$. Using this in equation (3), we obtain
$$\frac{(0.4 \times P_{S_p} + 0.6 \times P_{S_p}) \times t_p}{(0.9 \times 0.4 \times P_{S_p} + 0.6 \times P_{S_p}) \times t_\alpha} > 1 \tag{5}$$
Simplifying equation (5) gives
$$\gamma_E = \frac{t_p}{0.96 \times t_\alpha} \tag{6}$$
as the system level energy savings. In other words, equation (6) specifies how the *application level performance* ratio impacts the system level energy efficiency. Figures 3 (a), (b) and (c) show the percentage performance degradations when $M_{f1}$ is used instead of $M_p$. As an example, Q17, Q3, Q22, and Q18, $t_p \approx t_{f1}$, and gives $\gamma_E \approx 1.042$, corresponding to system level energy savings of 4.2%. All the queries, except Q1 and Q8 have $\frac{t_p}{t_{f1}} \approx 0.98$ to 1, and this corresponds to system energy savings of 2 to 4%. For Q1 and Q8 however, slowing the memory results in large performance drops. For Q1, $\frac{t_p}{t_\alpha} = 0.9$, resulting in an energy loss of 8%; Q8 has even larger performance and energy losses.

This clearly shows that most queries do not require the memory system to be operating at the peak frequency (with lowest possible latency). The key observation is that in many cases energy savings can be easily obtained (almost for free), even though the savings are only 2% to 4%. To achieve the savings, we can use a simple scheme to identify the queries that are sensitive to the memory's frequency by monitoring application performance over smaller time intervals, and delivering feedback to the memory controller.

*2) Reducing Memory Capacity ($\alpha$=$c1$, $c2$ or $c3$):* Reducing the memory capacity from $M_p$ to $M_\alpha$ scales the total power by a fraction $x$=0.75, 0.5 and 0.25 for $M_{c1}$, $M_{c2}$, and $M_{c3}$ respectively. Using this in equation (2), and diving by $P_{S_p}$ gives
$$\gamma_E = \frac{t_p}{(0.4 \times x + 0.6) \times t_\alpha + \frac{P_\Delta}{P_{S_p}} \times t_\Delta} \tag{7}$$
Let's consider $M_{c2}$, where $x = 0.5$. We now have
$$\gamma_E = \frac{1}{0.8 \times \frac{t_{c2}}{t_p} + \frac{P_\Delta}{P_{S_p}} \times \frac{t_\Delta}{t_p}} \tag{8}$$
To make a conservative estimate, we assume that $P_\Delta \approx P_{S_p}$. The value of $\gamma_E$ now depends on the performance ratios $\frac{t_{c2}}{t_p}$ and $\frac{t_\Delta}{t_p}$. Figures 3 (d), (e), (f) show the percentage performance degradations for the various queries. Q2, Q3, Q5, Q13, and Q22 have $\frac{t_{c2}}{t_p} \leq 1.04$. Substituting the upper bound in equation (8) gives $\gamma = \frac{1}{0.832+\frac{t_\Delta}{t_p}}$. The value of $t_p$ can be several minutes; if we assume that the overhead in reducing the capacity of memory is 5%, then the value of $\gamma$ is $\approx 1.14\%$, corresponding to system energy reduction of at least 14%. Three queries: Q9, Q17, and Q20 suffer performance degradations of 46%, 200% and 589% and cannot benefit from the lowered capacity.

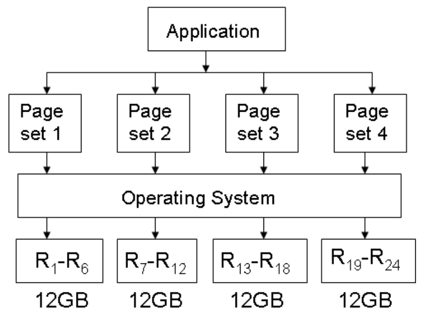
For $M_{c3}$, with $x = 0.25$, we have
$$\gamma_E = \frac{1}{0.7 \times \frac{t_{c3}}{t_p} + \frac{t_\Delta}{t_p}} \tag{9}$$
Most of the queries in Figures 3 (d), (e), (f) have $\frac{t_{c3}}{t_p} \leq 1.07$. Assuming the same 5% overhead, we can obtain $\gamma = 1/0.799 \approx 1.25$. This corresponds to platform energy savings of 25%. If the overhead in making the transition is 10%, the savings would be 15%. This is again a conservative estimate, since 10% of the execution time of a query may be several minutes, which is a large allowance for an overhead. In this case, for $M_{c3}$, four queries: Q9, Q22, Q17, and Q20 suffer large performance degradations and do not benefit from the reduced capacity. Similar analysis can be performed for $c1$ (reducing capacity by 25%) and $cf2$ (reducing capacity by 50% and reducing frequency to 866MHz).

The analysis presented in this section shows that it is possible to save up 15-25% system energy if the memory configuration can be made adaptive to workload behavior. Given the way existing memory systems operate, all 48GB of memory need to be powered for all queries *because of the few queries that require larger amounts of memory*. This is because the pages containing the data are interleaved across all memory ranks, and all 48GB of memory ranks would be "touched" by the application, even if the total data accessed may be much lesser than 48GB. The solution lies in adaptively compacting memory references over subsets of ranks. Enabling this involves several considerations. In the next section, we discuss some of the considerations involved, and present some motivating evidence.

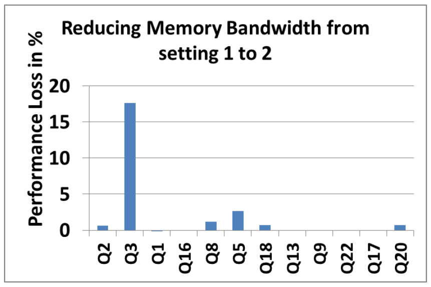
## IV. Discussion

Our observations indicate that adaptively adjusting memory configurations based on applications' behavior may provide significant energy savings. Adjusting frequencies is relatively simple and the overhead is negligible. Changing capacities, however, requires that the application change how it uses memory. This is a more complex problem; a solution would require the involvement of multiple layers across the software stack, including database applications, the operating system, and the chipset and memory controller. It would probably involve adaptively mapping subsets of pages onto subsets of ranks, as shown in Figure 5 (a), and possibly involve principles from prior work on power-aware page mapping.

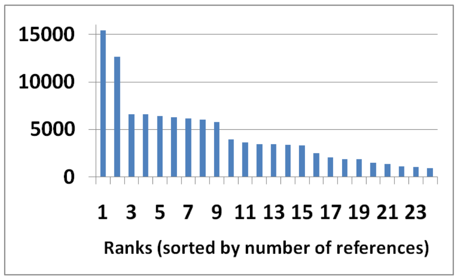
One consequence of power management using different memory configurations is that the effective bandwidth available to the application is reduced. When some pages are mapped onto a smaller set of ranks, successive references could be mapped onto a smaller set of memory controllers and ranks. This indicates a reduction in bandwidth. In the remainder of this section, we present some analysis on the performance of the DSS queries at a reduced bandwidth, and on the spatial distribution of some sampled memory references. Our analysis supports the hypothesis that the application does not suffer substantial performance degradation when the bandwidth is reduced, and shows that existing memory



(a)



(b)



(c)

Fig. 5. (a) Application-level, OS and chipset need to collaborate for compaction. Application would decide which sets of pages are important during compaction, and deliver performance feedback to the lower layers for a memory configuration. The OS and chipset would decide when and how to alter the memory configuration. (b) Reducing memory bandwidth from $S_1$ to $S_2$. It is observed that most of the queries do not incur much performance degradation at the lower bandwidth, indicating that it may not be necessary to always interleave memory references as much as possible. (c) Rank-wise distribution of memory references in a 0.5s interval: the x axis is sorted in the order of number of memory references. It is observed that there is a a skew in the distribution, indicating that ranks are not referenced uniformly.

references already exhibit a skew in their spatial distribution of references.

## A. Reducing Memory Bandwidth

| Setting | Memory Read | Memory Write |
|---|---|---|
| $S_1$ | 37479 MB/s | 36553 MB/s |
| $S_2$ | 15089 MB/s | 13531 MB/s |

TABLE IV
MEMORY BANDWIDTH SETTINGS

We reduce the server's memory bandwidth using MSRs, and obtain two settings ($S_1$ and $S_2$). We use a tool that measures the platform bandwidth with all prefetchers disabled, and find that the two settings have the bandwidths shown in Table IV. Figure 5 (b) shows the execution time of the queries, when the system bandwidth is reduced from $S_1$ to $S_2$. It is observed that most of the queries have very small drops in performance. This indicates that in most cases, the application can sustain its performance under lower bandwidths; this could enable less interleaved, more power efficient memory configurations.

## B. Memory Address Access Patterns

In this section, we examine the spatial distribution of the memory references made by the application. We collect spatial access patterns of the application using a hardware sampling tool. The patterns correspond to the virtual addresses of the application; examining these patterns provides an idea of how memory is spatially referenced in a given time window. Figure 5 (c) shows the rank-wise distribution of memory references in a time interval of 0.5 seconds. The rank-wise distribution is constructed using the linear addresses, since a distinct linear address typically corresponds to a distinct physical address. We observe that there is a spatial skew towards a subset of ranks. This skew in access distribution - of some ranks being utilized a lot more than others - persisted for two different address maps, and over different time intervals. This indicates that the ranks are not utilized equally by the application, and the under-utilization can be addressed by a compacting scheme that uses fewer ranks and is more energy efficient.

## V. CONCLUSION

This paper presents the power consumption in different scenarios on an enterprise server running a database program for decision support. The measured data suggest that there is large potential for energy savings, if the application can coordinate with the lower layers of the software stack, allowing the memory configuration to be adaptive. Many enterprise applications are already adaptive to the amount of available memory. As energy efficiency becomes an increasingly important factor in future server design, we envision that future software will incorporate power management into the design.

## REFERENCES

[1] M. Tolentino, J. Turner, and K. Cameron, "Memory miser: Improving main memory energy efficiency in servers," *IEEE Transactions on Computers*, pp. 336–350, 2008.
[2] Q. Deng, D. Meisner, L. Ramos, T. Wenisch, and R. Bianchini, "MemScale: Active Low-Power Modes for Main Memory," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2011.
[3] D. Meisner, B. Gold, and T. Wenisch, "PowerNap: eliminating server idle power," *ACM SIGPLAN Notices*, vol. 44, no. 3, pp. 205–216, 2009.
[4] M. Ware, K. Rajamani, M. Floyd, B. Brock, J. Rubio, F. Rawson, and J. Carter, "Architecting for power management: The IBM® POWER7 approach," in *HPCA*, 2010, pp. 1–11.
[5] http://www.postgresql.org/, "PostgreSQL."
[6] http://osdldbt.sourceforge.net/, "Open Source Database Labs."
[7] M. Bi, R. Duan, and C. Gniady, "Delay-Hiding energy management mechanisms for DRAM," in *International Symposium on High Performance Computer Architecture*, 2010, pp. 1–10.
[8] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *International Symposium on Computer Architecture*, 2009, pp. 24–33.
[9] S. Chahal and T. Glasgow, "Memory Sizing for Server Virtualization," in *White Paper*. Intel Corporation, 2007.
[10] A. Dhodapkar and J. Smith, "Comparing program phase detection techniques," in *IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2003, pp. 217–227.
[11] A. R. Lebeck, X. Fan, H. Zeng, and C. Ellis, "Power aware page allocation," *ACM SIGOPS Operating Systems Review*, vol. 34, no. 5, pp. 105–116, 2000.
[12] H. Huang, P. Pillai, and K. Shin, "Design and Implementation of Power-aware Virtual Memory," in *USENIX Annual Technical Conference*, 2003, pp.5–16.
[13] http://www.tpc.org, "Transaction Processing Council."