Is Storage Hierarchy Dead? Co-located Compute-Storage NVRAM-based Architectures for Data-Centric Workloads

David Roberts, Jichuan Chang, Parthasarathy Ranganathan, Trevor N. Mudge

HP Laboratories
HPL-2010-119

Abstract:

The increasing gap between the speed of the processor and the time to access the data in the disk has historically been offset with deeper and larger memory hierarchies with multiple levels of SRAM, DRAM, and more recently, Flash layers for caching. However, recent trends that point to a potential slowdown of DRAM growth and the emergence of alternate resistive non-volatile memory technologies and properties of emerging data-centric workloads offer the opportunity to rethink future solutions. Specifically, in this paper, we examine an approach that leverages both the memory-like and disk-like attributes of emerging non-volatile memory technologies. We propose a new architectural building block - called nanostores - that co-locates computation with a single-level data store in a flat hierarchy, and enables large-scale distributed systems for future data-centric workloads. We present a new evaluation methodology to reason about these new architectures, including benchmarks designed to systematically study emerging data-centric workloads. Our evaluation results demonstrate significant potential for performance benefits from our approach (often orders of magnitude) with better energy efficiency.

External Posting Date: November 8, 2010 [Fulltext] Approved for External Publication
Internal Posting Date: November 8, 2010 [Fulltext]

© Copyright 2010 Hewlett-Packard Development Company, L.P.

Is Storage Hierarchy Dead? Co-located Compute-Storage NVRAM-based Architectures for Data-Centric Workloads

The increasing gap between the speed of the processor and the time to access the data in the disk has historically been offset with deeper and larger memory hierarchies with multiple levels of SRAM, DRAM, and more recently, Flash layers for caching. However, recent trends that point to a potential slowdown of DRAM growth and the emergence of alternate resistive non-volatile memory technologies and properties of emerging data-centric workloads offer the opportunity to rethink future solutions. Specifically, in this paper, we examine an approach that leverages both the memory-like and disk-like attributes of emerging non-volatile memory technologies. We propose a new architectural building block – called nanostores – that co-locates computation with a single-level data store in a flat hierarchy, and enables large-scale distributed systems for future data-centric workloads. We present a new evaluation methodology to reason about these new architectures, including benchmarks designed to systematically study emerging data-centric workloads. Our evaluation results demonstrate significant potential for performance benefits from our approach (often orders of magnitude) with better energy efficiency.

1. INTRODUCTION

The amount of data being created is exploding, growing significantly faster than Moore's law. For example, the size of the largest data warehouse in the Winter Top Ten Survey has been increasing at a cumulative annual growth rate of 173% [34]. The amount of online data is estimated to have risen nearly 60-fold in the last seven years [24]. Data from richer sensors, digitization of offline content, and new applications like twitter, search, etc., will only increase data growth rates. Indeed, it is estimated that only 5% of the world's offline data has been made online so far [22].

This growth in data is leading to a corresponding growth in data-centric applications that operate on data in diverse ways (capture, classify, analyze, process, archive, etc). Compared to traditional enterprise workloads (e.g., online transaction processing, web services), emerging data-centric workloads change a lot of assumptions about system design. These workloads typically operate at larger scale (hundreds of thousands of servers) and on more diverse data (e.g., structured, unstructured, rich media) with I/O intensive, often random, data access patterns and limited locality. In addition, these workloads have been characterized by a lot of innovations in the software stack targeted at increased scalability and commodity hardware (e.g., Google MapReduce/BigTable).

Concurrent with these trends, significant changes are also expected in the memory industry. Recently, new non-volatile RAM (NVRAM) memory technologies have been demonstrated that significantly improve latency and energy efficiency compared to Flash and Hard Disk. Some of these NV memories, such as Phase-Change Memory (PCM) and Memristors have been demonstrated to have the potential to replace DRAM with competitive performance and better energy efficiency and technology scaling. At the same time, several studies have postulated the potential end of DRAM scaling (or at least a significant slowing down) [4] [18] [27] [39] over the next decade, further increasing the likelihood of DRAM being replaced by these NVRAM memories in future systems.

The confluence of these trends – future large-scale distributed data-centric workloads with I/O intensive behavior, the corresponding innovations in the software stack, the end of scaling for DRAM, and their

potential replacement with NVRAM memories – offers a unique opportunity to rethink traditional system architecture and memory hierarchy design for future workloads. Specifically, in this paper, we present a radical (yet intuitive) new design that co-locates computing with non-volatile storage, eliminating many intervening levels of the memory hierarchy. All data is stored in a single NVRAM data-store layer that replaces traditional disk and DRAM layers (disk is relegated to archival backup).

The rest of the paper is organized as follows. Section 2 motivates and presents the design of *nanostores*, a new building block for data-centric systems. A nanostore includes 3D-stacked non-volatile memory with a layer of compute cores and a network interface, and can operate as a system node in a larger distributed system running a data-parallel environment like MapReduce. Section 3 presents a new evaluation methodology to reason about these new architectures. Our approach includes a hybrid evaluation model that incorporates high-level application models akin to database query plans or MapReduce simulations in combination with detailed micro-architectural simulations and a data-centric workload taxonomy and a benchmark suite designed to systematically exercise this taxonomy. Section 4 presents our evaluation results demonstrating significant benefits even when compared to aggressive future extrapolations of current best systems – one to three orders of magnitude performance improvements at 2X to 10X improved energy efficiency. We break down the benefits across the key design aspects, discuss workload-specific trends and key assumptions in leveraging these benefits, and present sensitivity results to technology trends and limits. Section 5 discusses related work and Section 6 concludes the paper.

2. ARCHITECTURE: NANOSTORES

2.1 Motivation

2.1.1 Data-centric workloads

An important trend in the emergence of data-centric workloads has been the emergence of complex analysis at immense scale (coupled closely with the growth of large-scale internet web services). Traditional data-centric workloads like web serving and online transaction processing are being superseded by workloads like real-time multimedia streaming and conversion, history-based recommendation systems, searches of text, images and even videos, and deep analysis of unstructured data (e.g., Google Squared).

From a system architecture point of view, a common characteristic of these workloads is that they are generally implemented on highly distributed systems and adopt approaches that scale by partitioning data across individual nodes. Their large scale is reflected both in the total amount of data involved in a single task and the number of distributed compute nodes required to process the data. Additionally, these workloads are I/O intensive often with random access patterns to small-sized objects over large data sets. Many of these applications are also operating on larger fractions of data in memory. A recent study reports that, for non-image data, the total amount of DRAM used in Facebook is approximately 75% of the total data size [11]. While this trend partly reflects the little or no locality due to complex linkages between data for the Facebook workload, similar trends can be seen for memcached servers and TPC-H winners over the past decade.

Similarly, search algorithms (e.g., from Google) have evolved to store their search indices entirely in DRAM. These trends motivate us to rethink the balance between memory and disk-based storage in traditional designs.

Second, recent data-centric workloads have also been characterized by a lot of commercially deployed innovations in the software stack (e.g., Google BigTable and MapReduce, Amazon Dynamo, Yahoo PNUTS, Microsoft Dryad, Facebook Memcached, LinkedIn Voldemort). Indeed, a recent talk mentions that the software stack behind the very successful Google search engine was re-architected significantly four times in the last seven years, to achieve better performance at increased scale [14]. The growing importance of this class of workloads, their focus on large-scale distributed systems with ever increasing use of memory, and their openness to software-level innovations together offer an opportunity for a corresponding clean-slate architecture design targeted at these workloads.

2.1.2 Technology trends

Concurrently, recent trends point to several potential technology disruptions in the horizon. On the compute side, recent microprocessors have favored multicore designs emphasizing multiple simpler cores for greater throughput. This is well matched with the large-scale distributed parallelism discussed earlier in data-centric workloads. Operating cores at near-threshold voltage has been shown to significantly improve energy efficiency [37]. Similarly, recent advances in networking, particularly around optics, show a strong growth in bandwidth for communication between different compute elements at various levels of the system design.

The most important technology changes pertinent to data-centric computing, however, relate to the advances and adoption of non-volatile memory. Flash memories have been increasingly widely adopted in popular consumer systems (e.g, Apple's iPhone) and are starting to gain adoption in the enterprise market (e.g, FusionIO). Emerging non-volatile memories have been demonstrated with superior properties to Flash, most notably, Phase-Change Memory (PCM) and, more recently, Memristors. Table 1 summarizes key attributes (density, bandwidth, latency, energy, retention, and endurance) of potential storage alternatives in the next decade, with projected data from recent publications and technology trends [4] [15] and direct industry communication. These trends suggest that future non-volatile memories can be viable DRAM replacements, achieving competitive speeds at much lower power consumption, and with non-volatility properties similar to disk but without the power overhead. Additionally, several recent studies have identified a slowing of DRAM growth ([4] [18] [39] [27]) due to scaling challenges for charge-based memories. The adoption of NVRAM memories as DRAM replacement can potentially be accelerated due to such limitations in scaling DRAM.

Technology	Density	Bandwidth	Rd Latency	Wr Latency	Rd Energy	Wr Energy	Endurance
	(µm^2/bit)	(GB/s)	ns	ns	pJ/bit	(pJ/bit)	Wr/bit
HDD	0.00006	0.5	3,000,000	3,000,000	2,500	2,500	N/A
SSD (SLC)	0.00210	1	25,000	200,000	250	250	1.0E+05
DRAM (DIMM)	0.00380	26	55	55	24	24	N/A
PCRAM (3D)	0.00580	32	48	150	2	19.2	1.0E+07
Memristor (3D)	0.00480	32	100	100	2	2	1.0E+07

Table 1: Storage technology projections for 2015 timeframe

Two traditional limitations of NVRAM technologies have been around density and endurance, but recent trends suggest that these limitations can be addressed. Increased density can be achieved within a single-die

through multi-level designs, and potentially multiple layers per die [19]. At a single chip level, 3D die stacking using through-silicon vias (TSVs) for inter-die communication can further increase density. Such 3D stacking also has the additional advantage of closely integrating the processor and memory for higher bandwidth and lower power (due to short length low capacitance wires). In terms of endurance, compared to flash, PCM and Memristor offer significantly better functionality (10^7-10^8 writes per cell compared to the 10^5 writes per cell for Flash). Optimizations at the technology, circuit, and systems levels [18] [39] [26] [27] have been shown to further address endurance issues, and more improvements are likely as the technologies mature and gain widespread adoption. (We discuss the lifetime of our proposed designs further in Section 4.)

These trends suggest that technologies like PCM and Memristors, especially when viewed in the context of advances like 3D die stacking, multicores, and improved networking, can induce more fundamental architectural change for data-intensive computing than traditional approaches that use them as solid-state disks or as another intermediate level in the memory hierarchy.

2.2 Proposed architecture

2.2.1 High-level design

Combining the opportunity for a clean-slate architectural redesign for data-centric workloads with the potential to use emerging NVRAM memories in more disruptive ways, we propose a new system architecture and memory system design that co-locates power-efficient compute cores with non-volatile storage, eliminating many intervening levels of the memory hierarchy. All data is stored in a single NVRAM data-store layer that replaces traditional disk and DRAM layers (disk is relegated to archival backup).

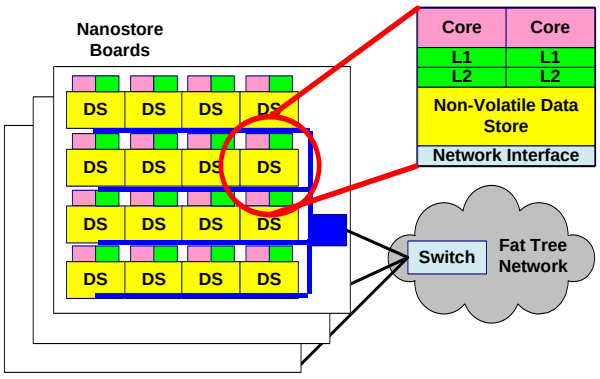


Figure 1: Nanostore system architecture

Figure 1 illustrates the proposed solution consisting of a distributed system comprised of a large number of small homogeneous building blocks. Our individual building block is a single chip and we refer to it as *nanostore*. A nanostore consists of multiple 3D-stacked layers of dense silicon non-volatile memories (e.g., PCM or Memristors) with a top layer of power-efficient compute cores. TSVs are used to provide wide, low-energy data paths between the processors and data stores. Power and thermal issues are important concerns with 3D-stacking. This limits the amount of compute that can be included in a nanostore. In this paper, we make 3D-packaging assumptions similar to the PicoServer project [12] and focus on low-power simpler cores for the nanostore. Our assumption of simpler cores also means that we can correspondingly provision the bandwidth to save power. Each nanostore can act as a full-fledged system with a network interface.

The individual nanostores are networked via an on-board connector to form a large-scale distributed system or cluster akin to the existing solutions for data-centric workloads discussed in Section 2.1.1. In terms of physical

organization, multiple nanostore chips are organized into small daughter boards that, in turn, plug into traditional blade server boards.

2.2.2 Specific choices

While the above description summarizes the high-level organization, there is a wide range of possible implementations. There are a number of design choices in terms of the provisioning, organization, and balance of the compute, storage, and network per nanostore as well as the sharing model across the individual nodes and the topology of the network (including potential differences between the on-chip, on-board, and cross-cluster networks). The design choices are constrained by technology- and circuits-level parameters such as the size of the die and the yield, or the number of 3D-stacked or intra-die layers, as well as packaging constraints such as the power/thermal budget per chip or board.

We assume a nanostore die size of 100mm^2, similar to the cost-effective design point for memories[1]. For a Memristor-based design circa 2015 [33], assuming 8 layers of 3D and intra-die stacking, the on-chip datastore capacity is 75 GB per socket, with 100ns access latencies and 2 pJ/bit access energy. An equivalent PCM-based design would have 25GB per socket with 150ns latencies and 2/19.2 pJ/bit read/write energy (Table 1). In addition to these two data points, we also study other parameters for data store capacity, latency, and energy to understand the sensitivity of our results to alternate NVRAM instantiations in the future.

The cores in the compute layer are based on low-voltage power-efficient microarchitectures with simple SRAM cache hierarchies. Different organizations are possible for the compute layer – in the number of cores (1 to 128), clock frequency (100MHz to 2GHz), issue width and pipeline depth (2-way simple to 4-way deep), and L2 cache size (512KB or 1MB per core) – and we study this design space. To ensure realistic designs, we limit the power density at the socket (32 Watt/cm^2). For our projected timeframe, we expect 3D stacking to provide significant bandwidth (up to 32GB/s per the PicoServer design [12]) between the processor and stacked memory, and 80Gbps (2x 40Gbps NICs) networking[2] bandwidth per system (in a traditional architecture) but we study these as design space parameters too. We assume a large-scale distributed shared-nothing system abstraction. This is well-matched with current data-centric workloads. Each nanostore can be viewed as a complete independent system executing the software stack needed to implement a data parallel execution environment like MapReduce.

2.2.3 Discussion

The two most important aspects of nanostores are (1) the co-location of power-efficient computing with a single-level data store, and (2) the support for large-scale distributed design. Together, these enable several

[1] Alternatively, we could have assumed 400mm^2 nanostore die sizes similar to that for processors, but we chose to be conservative. (A larger die size means higher storage density and compute-capacity per nanostore and potentially reduced scalability/networking limitations for our design.)

[2] We limit our discussion in this paper to a fat-tree network topology with the system nodes as the leaves and the switches as the root nodes. However, our proposed design should work well (or better) with other topologies including recently proposed approaches like HyperX.

benefits. The single-level data store enables improved performance due to faster data access (in latency and bandwidth). Energy efficiency is also improved from the flattening of the memory hierarchy and the increased energy efficiency of NVRAM over disk and DRAM. The large-scale distributed design allows for higher performance from increased parallelism and higher overall data/network bandwidth. This design also improves energy efficiency by partitioning the system into smaller elements that can leverage more power-efficient components (e.g., simpler cores).

At the same time, there are potential disadvantages. Given the smaller capacities of per-socket storage, the number of individual elements in the system increases dramatically. This can potentially increase the stress on the networking subsystem – around bandwidth contention (particularly for all-to-all communication), topological complexity and port count, and power. Software scalability can also be an issue. While large-scale deployments of data-centric workloads have been demonstrated, latency requirements (e.g., 500ms response time for a search request) will still have to be carefully considered in the sizing of the system. Finally, chip-level thermal constraints can limit the compute per nanostore; this could lead to compute bottlenecks (e.g., for sophisticated data processing like collaborative filtering).

This discussion points to several open questions. How well does our nanostore design perform compared to aggressive extrapolations of existing approaches? Are the expected benefits significant enough to warrant the change? How do the benefits change across the range of data-centric workloads? How do the benefits break down? Do we need to rethink the balance of compute, data, and network for this new architecture? What are the implications of specific design choices and technology extrapolations? In particular, what is the sensitivity to the network bandwidth assumptions and packaging limitations? What are the implications of limited endurance of non-volatile memories? What are the implications for future research? The rest of the paper seeks to address these questions.

3. EVALUATION METHODOLOGY

3.1 Challenges

Evaluating our proposed architecture and understanding the design space poses several challenges. Our focus on the combination of multiple future technologies for emerging workloads poses several challenges in the choice of benchmarks, technology parameters and baseline systems appropriate to this longer timeframe. Furthermore, irrespective of any specific assumptions we make, given the speculative nature of such an exercise, we need to relate these assumptions to a systematic understanding of the trends as well as understand the variance of the results to alternate futures (affecting both the specific parameters and the combinations in which they are used).

To evaluate our proposed design and its tradeoffs, we need to study large-scale clusters running distributed workloads operating on large volumes of data. We also need to examine tradeoffs at the full system level including computing, networking, memory, and storage layers. Conventional architecture simulators not only lack the ability to cope with this level of system scale, but also the modeling means for storage and network subsystems at a distributed systems level. There is also a combinatorial explosion in the design space from various assumptions at the fine-grained and coarse-grained architectural levels as well as the choice of technology and workload parameters. An appropriate evaluation methodology is required to systematically reason about this large design space.

To address these challenges, we next discuss a new set of benchmarks that provides systematic coverage of data-centric workloads and develop a new evaluation framework that uses hybrid performance models to reason about future new architectures.

3.2 Proposed benchmarks

The space of data-centric workloads is vast, fast-evolving, and characterized by rich diversity across multiple dimensions. To study a subset of workloads that provide sufficient coverage and representativeness, we systematically create a taxonomy of data-centric workloads to characterize the key dimensions of diversity and pick a subset of workloads that exercises all these dimensions.

Response Time	Real-time	Real-time or interactive responses required
	Background	Response time is not critical for user needs
Access Pattern	Random	Unpredictable access to regions of data store
	Sequential	Sequential access of data chunks
	Permutation	Data is re-distributed across the system
Working Set	All	The entire dataset is accessed
	Partial	Only a subset of data is accessed
Data Type	Structured	Metadata/schema/type are used for data records
	Unstructured	No explicit data structure, e.g., text/binary files
	Rich media	Audio/video and image data with inherent structures and specific processing algorithms
Read vs. Write	Read heavy	Data reads are significant for processing
	Write heavy	Data writes are significant for processing
Processing Complexity	High	Complex processing of data is required per data item. Examples: video trans-coding, classification, prediction
	Low	Dominated by data access with low compute ratio. Examples: sort, upload, download, filtering, and aggregation.

Table 2: A data-centric workload taxonomy

	Access: Random	Access: Sequential	Access: Permute	Time: Real-time	Time: Background	Compute: High	Compute: Low	Write-Heavy	Read-Heavy	Working Set: All	Working Set: Partial	Data: Structured	Data: Unstructured	Data: Rich media
Sort			X		X		X	X		X		X		
Search (web)	X			X		X			X	X		X		
Search (image)	X			X		X			X	X			X	X
Search Indexer	X				X	X			X	X		X	X	X
Recommender	X				X	X			X	X		X		
De-duplication		X			X		X		X	X		X	X	
Transaction	X	X	X	X			X	X			X	X		
Decision Support	X	X	X		X	X			X		X	X		
Video Sharing	X	X		X	X	X			X		X	X	X	X
Data Mining			X		X		X		X	X		X		

Table 3: Workload mapping

Table 2 illustrates the taxonomy of data-centric workloads that we developed based on examination of a wide class of emerging applications. Key dimensions include: *response time* (real-time vs. background), *access pattern* (random, sequential or permutation), *working set* (all vs. partial), *data type* (structured, unstructured and rich media), *type of access* (read vs. write dominated), and *processing complexity* (low, medium or high). Table 2 further explains the attributes of each dimension. Table 3 shows an example of mapping some popularly-referenced workloads to the taxonomy and picking a small subset with full coverage (shaded rows). Our chosen workloads provide representative coverage of different dimensions of data-centric workloads, capture emerging trends towards data analysis and

media processing, and support publicly available implementations that we can use for simulation. We describe these workloads next.[3]

Sort: The *sort* benchmark models a two-pass distributed sort of a 1 petabyte dataset. The workload is both read- and write-heavy, and stresses the balance between compute/storage/networking subsystems. The algorithm we study is massively data-parallel and has two phases: (i) a *shuffle* phase where each server first reads keys from its local storage and sends them over the network to their individual target servers while simultaneously receiving keys from other servers. As the memory fills with incoming keys, the server sorts the buffered keys and writes the results to its local storage. (ii) a *local merge* phase after the shuffle, where each nanostore reads partially sorted keys from many local files, performs a merge sort then writes the final sorted results to local storage. We use the `nsort` benchmark to model the local sort phases of this workload.

Checksum in data deduplication: This benchmark *cksum* implements checksum calculation in data deduplication, resulting in mostly read-only, sequential access with low processing complexity. We model a massively parallel implementation of *cksum* over a 1 petabyte dataset. Each server scans its local files and generates block or file signatures using the SHA-1 hash function. We use `sha1sum` for our simulations.

Video transcoding: This benchmark models popular video encoding/transcoding web services that exploit cloud infrastructure for batch processing. The algorithm reads the video input files, transcodes, and then stores the output in a new format locally. For our simulations, we use the `ffmpeg` benchmark over a 1 petabyte dataset. Our sample video is in FLV format at 320x240 resolution and is transcoded into JPEG snapshots. The average size of the video is 10MB.

Recommender: This benchmark *recom* represents parallel machine learning algorithms with high processing complexity and regular communication patterns. Our workload models the Netflix video recommendation benchmark [40] over a 5 Terabyte dataset using parallel matrix factorization. The algorithm iteratively refines two matrices so their product can best summarize the *ratings* matrix. The large matrices are distributed across the servers' main memory. Each iteration has four phases, two of them are compute-heavy matrix operations while in the other interleaving phases. The servers update their local copies of input matrices using personalized broadcast messages. The implementation requires large main memory to host the matrices and compute/communication balance. For our simulations, we use a matlab compiled winner of the `Netflix` challenge from 2008.

Search: The *search* benchmark models text search across a 128 terabyte data set, using an in-memory index to achieve sub-second response times. The workload is read-only and stresses random access patterns. Similar to popularly-used in-memory index-based text searches (e.g., Google), the entire index is partitioned and stored in-memory in a large-scale distributed cluster. Each server searches its local index in the *map* phase, and sends the top-matching document list to the front-end server in the *reduce* phase. In addition to search query

[3] We hope to release the benchmarks and the execution models as a web appendix with the paper.

throughput, this benchmark models a quality of service (QoS) requirement that average query latency should be less than 500ms. For our simulations, we use the `lucene` benchmark for the map phase.

3.3 Proposed evaluation methodology

3.3.1 Two-level simulation with design optimizer

The evaluation methodology that we use is summarized in Figure 2. Specifically, our approach has three main components: (1) a high-level distributed system model, (2) a lower-level microarchitecture-based model, and (3) a design space optimizer.

Our high-level distributed system simulation captures the applications' system-level behavior and allows exploration of broader datacenter issues like topology and compute-network-datastore balance. Our implementation is inspired by approaches currently used for mapreduce/Hadoop simulation (e.g., [36]) and query optimizers in databases (e.g., [35]). Similar to these approaches, our high-level model uses an application-level execution template that breaks down execution time into key compute, communicate, and I/O subsystem phases coupled with models for performance and power for each subsystem. Given the per-node focus of the nanostore in this paper, we use high-level models for the network and I/O subsystems, but use the input from a detailed microarchitectural simulator to model the compute subsystem. This lower-level microarchitecture-based model captures the applications' instruction-level behavior and allows exploration of architecture choices like ILP and cache hierarchies. The compute data throughput and memory bandwidth results from this model feed into the higher-level simulation. A final design space optimizer iterates between various compute, storage, and network options to choose the optimal balanced design for a given objective function (energy-delay product, energy efficiency, or performance). Figure 3 presents the execution template and the high-level models for the *sort* benchmark as an illustrative example.

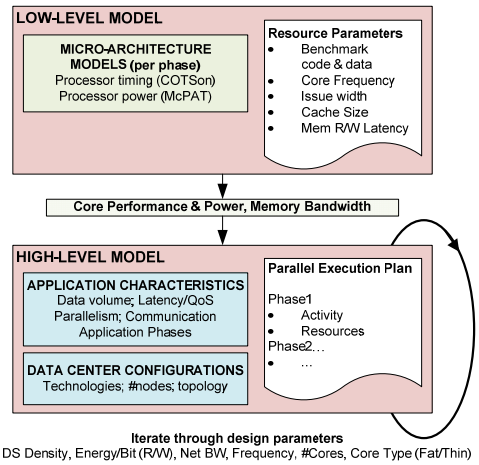


Figure 2: Two-level simulation with design optimizer

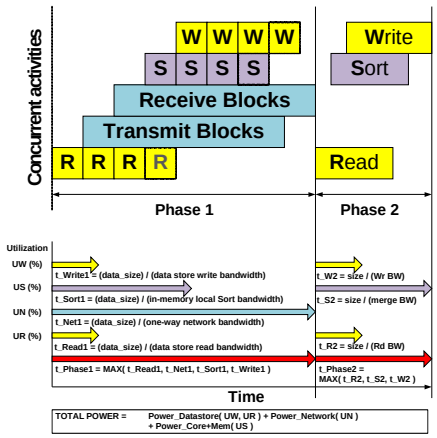


Figure 3: Illustration of *sort* execution template

Our evaluation methodology is unique in addressing the challenges discussed earlier in evaluating a new architectural model, with forward-looking technology and workload assumptions, on large-scale distributed

systems. It also addresses all the compute, I/O, and network components of the system, and provides a powerful way to systematically explore the rich design space at practical simulation times. However, a few caveats need to be noted. Our high-level application execution models assume that computation and network communication can overlap and are purely bandwidth based (i.e. no queuing models are used). These assumptions are acceptable for the distinct phases and coarse-grained communication behavior of the representative benchmarks we consider in this study, but care needs to be exercised in extrapolating the model to other workload classes. Data is also assumed to be distributed uniformly, and load-balancing effects are ignored. For the purposes of our study, these assumptions allow an adequate comparison of hardware architectures under reasonable software conditions. However, issues around optimizing the operating systems and middleware for load-balancing and scheduling need to be addressed in future work. Finally, in the absence of actual prototypes (which is difficult given that many of the technologies we study are still lab samples), it is hard to validate the model for futuristic design configurations. However, for simple near-term design alternatives, we have tried to make sure that our results match prior published trends. For example, we studied the performance of a simple MPI-based sort implementation on small cluster sizes (4 to 64 cores) and found the results tracked the model well. In addition, we validated all the execution templates with application experts for the respective benchmarks, and try to validate the intuition for all results using backup statistics where possible.

3.3.2 Performance and power models

The data store and network subsystem performance models calculate the execution time for storage access and communication activities based on the provisioned bandwidths of the subsystems and the amount of data transferred to and from the subsystem for the workload execution. For the data store bandwidth, we model the combined bandwidth needs of both file and memory accesses in our proposed designs. For the compute subsystem, we use the publicly-available benchmarks discussed earlier for each of our workloads and proceed in two steps. We execute the benchmark on an existing Xeon-based server with the baseline processor and DRAM. The server is configured with minimal storage and network overhead (e.g., in-memory search and processing cached data files). This experiment allows us to measure a “compute data processing throughput” baseline. For different specific processor configurations, we run the benchmark through a detailed publicly-available microarchitectural simulator (COTSon [3]) and use the simulated IPC values to normalize the data throughput rate.

For our results in the paper, we focus mainly on average power consumption. (We also study peak power consumption, but use it primarily to verify compliance with cooling constraints.) We use the execution time models to compute the utilization for the processors, memory, data store and network ports. Active power is assumed to scale linearly up to peak power as utilization increases. Several components also have a non-zero idle power (such as DRAM refresh or CPU leakage power). During phases where the CPU cores are active, we use the memory bandwidths (read and write) simulated in COTSon to calculate the memory utilization.

Since network packets traverse multiple hops in the datacenter, we scale our NIC-level power model with a network layer multiplier (the effective levels in a fat tree switch network) to calculate the total network power. The CPU peak power is determined as a function of issue width, frequency and cache size using [20]. CPU configurations at each frequency also factor in the power benefits from voltage and frequency scaling using the models specified in [20]. CPU idle power is scaled based on the number of cores and caches on the nanostore.

3.4 Choice of parameters and baselines

As discussed earlier, we examined ITRS roadmaps and prior publications and talked to industry sources to determine forward-looking parameters for our studies, but in most cases also study sensitivity to a range of values for each parameter. Also, we have generally tried to choose parameters that favored more conservative projections for the benefits of our proposal.

Table 4 summarizes our system parameter assumptions. The different compute configurations (varying number of thin/fat cores with different cache sizes), nanostore memory configurations (memristor/PCM), and data/network bandwidth assumptions that we already discussed in Section 2 are listed. For DRAM, we ignore any potential end-of-life scaling limitations at our projected time frame and extrapolate historical scaling trends in capacity and bandwidth. We assume configurations of 16GB per DRAM DIMM module and 25.6 GB/s bandwidth, with each DIMM consuming 10W at peak and 2W at idle. For persistent storage, we assume HDDs at 6TB capacity, 500 MB/s bandwidth, and active power consumption varying between 8W to 10W from idle to peak. We also study SSDs at lower capacity per drive (1.2TB), but higher bandwidth (4.5GB/s) and improved energy efficiency (10W peak power and 1W idle power). For the networking subsystem, we assume peak and idle power for a 40Gig Ethernet NIC of 10W and 2W respectively. The changes to the network topology corresponding to the increase in the number of leaf nodes with the relatively small capacity networked nanostores is modeled as an increase in the number of layers in the topology.

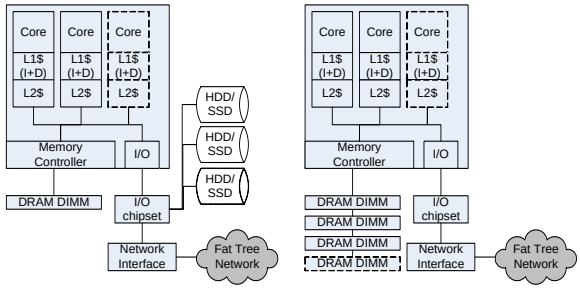
Processor	Baseline	Nanostore
Core count	32	1-128
Frequency (GHz)	2	0.1-2.0
Issue width	4	2, 4
Per-core L1 cache	64K+64K	64K+64K
Per-core L2 cache	1M	512K, 1M
Peak Power/Core (W)	1.83	(model)
Idle Power/Core (W)	0.04	(model)

Main Memory	Baseline	Nanostore
Peak BW/Unit (GB/s)	25.6	32
Capacity/Unit (GB)	16	25, 75
Peak Power/Unit (W)	10	0.6, 0.3
Idle Power/Unit (W)	2	0.0, 0.0

Network	Baseline	Nanostore
Peak BW/Port (Gbit/s)	40	40
Peak Power/Port (W)	10	10
Idle Power/Port (W)	2	2

Hard Disk/SSD	HDD	SSD
Peak BW/Drive (GB/s)	0.5	4.5
Capacity/Drive (TB)	6	1.2
Peak Power/Drive (W)	10	10
Idle Power/Drive (W)	8	1

Table 4: System parameters



(a) Sort, cksum, video (b) search, recommender

Figure 4: Baseline architectures

Figure 4 summarizes the baseline system architectures we study for each workload. To provide a fair comparison to the baseline, we recognize that different sweet spot design configurations have evolved for different workloads (in terms of emphasis on compute, storage, and networking, and organization), and

correspondingly choose a baseline known to be best-suited for each workload. Specifically, *sort*, *cksum* and *video* benchmarks keep their data on disks and are each allocated a single DRAM DIMM. Search and Netflix are in-memory workloads so do not have a hard disk component, but have multiple DIMMs. To ensure the most appropriate balanced baseline, other than the choice of an enterprise-class processor, other design parameters in the baseline are determined through the iterative design space search of our simulator.

4. EVALUATION RESULTS

4.1 Baseline benefits

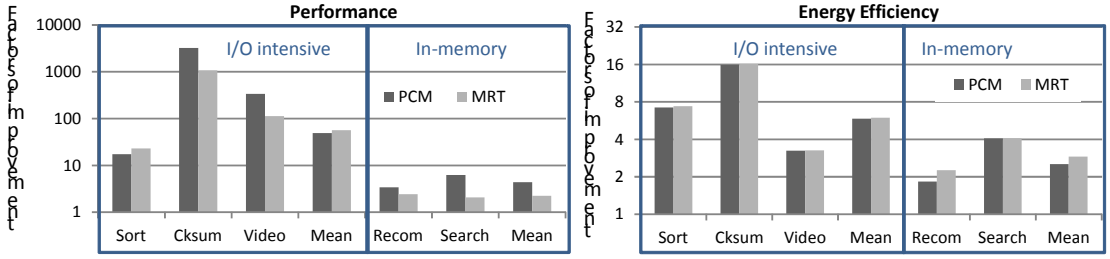


Figure 5: Performance and energy efficiency improvements over 2015 baselines (EDP-optimized)

Figure 5 presents the improvements in performance and energy efficiency from our nanostore designs relative to the baselines discussed above. Results for both the PCM- and Memristor-based nanostore designs are presented for the five benchmarks. For these results, consistent with the data-centric focus of this paper, we assume that the baseline and nanostore systems both operate on the same dataset size and keep the amount of permanent persistent storage the same. (The data size and device capacity together determine the number of sockets in the nanostore designs; all other parameters are based on Section 4.) Also, as discussed earlier, each point represents the results of a design space search by the optimizer across a range of configuration parameters, for both the baseline and nanostore designs, but with caps on the thermal density and aggregate network bandwidth as discussed in Section 2.2. This ensures that the individual designs are locally balanced for their objective function while meeting the design constraints. We focus on energy-delay-product as the primary objective function (since we want to optimize both energy efficiency and performance) but discuss other objective functions briefly in Section 4.3.

The results in Figure 5 show that for all our benchmarks, the nanostore solutions achieve higher performance at better energy efficiency. For the three I/O intensive benchmarks – *sort, cksum, video* – the nanostore designs achieve one to three orders of magnitude higher performance improvement with 3X-16X improved energy efficiency. For the in-memory benchmarks with DRAM baselines – *recom, search* – nanostores achieve 2X-6X better performance with 2X-4X improved energy efficiency.

Comparing the two different NVRAM technologies we consider, the PCM-based nanostores generally outperform the memristor-based designs, but at reduced energy efficiencies. However, it should be noted that our constant dataset size constraint presents the memristor-based design at a disadvantage. While, as the results indicate, a pre-packaged 1 petabyte PCM-based design would have more performance (and cores) than

a pre-packaged 1 petabyte memristor-based design, the PCM-based design also has more individual nanostore sockets (and correspondingly more volume). If an alternate comparison considered performance for the same number of sockets (or equivalently the same amount of silicon), the memristor-based design would have higher performance than PCM (by a factor corresponding to the *Node* multiplier discussed later).

4.2 Analysis of performance benefits

Bench	Scheme	1/EDP	Perf	EE	Configuration	ScaleX (Node/OPS/DS/Net)			
Sort	Base	1.0	1.0	1.0	Core=28/Freq=2.0/Issue=4/L2=1024k/DS=6000GB/nDS=2/DS_BW=0.5GBs/Net_BW=0.500GBs	1	1	1	1
	SSD	47.2	15.3	3.1	Core=128/Freq=2.0/Issue=2/L2=1024k/DS=1280GB/nDS=2/DS_BW=4.5GBs/Net_BW=1.250GBs	5	11	42	12
	DRAM	20.9	20.5	1.0	Core=128/Freq=0.5/Issue=2/L2=512k/DS=16GB/nDS=16/DS_BW=1.6GBs/Net_BW=0.125GBs	47	27	1198	12
	PCM	124.8	17.3	7.2	Core=22/Freq=0.1/Issue=2/L2=512k/DS=25GB/nDS=1/DS_BW=32.0GBs/Net_BW=0.013GBs	479	9	15329	12
	MRT	170.8	23.1	7.4	Core=88/Freq=0.1/Issue=2/L2=512k/DS=75GB/nDS=1/DS_BW=32.0GBs/Net_BW=0.050GBs	160	13	5110	16
Cksum	Base	1.0	1.0	1.0	Core=40/Freq=2.0/Issue=4/L2=1024k/DS=6000GB/nDS=14/DS_BW=0.5GBs/Net_BW=0.001GBs	1	1	1	1
	SSD	316	42	7.5	Core=104/Freq=0.5/Issue=4/L2=512k/DS=1280GB/nDS=2/DS_BW=4.5GBs/Net_BW=0.001GBs	33	21	42	1
	DRAM	3695	583	6.3	Core=128/Freq=1.0/Issue=2/L2=1024k/DS=16GB/nDS=16/DS_BW=1.6GBs/Net_BW=0.001GBs	326	263	1198	1
	PCM	51353	3235	15.9	Core=128/Freq=0.5/Issue=2/L2=512k/DS=25GB/nDS=1/DS_BW=32.0GBs/Net_BW=0.001GBs	3333	1344	15329	1
	MRT	17432	1078	16.2	Core=128/Freq=0.5/Issue=2/L2=512k/DS=75GB/nDS=1/DS_BW=32.0GBs/Net_BW=0.001GBs	1111	448	5110	1
Video	Base	1.0	1.0	1.0	Core=40/Freq=2.0/Issue=4/L2=1024k/DS=6000GB/nDS=2/DS_BW=0.5GBs/Net_BW=0.001GBs	1	1	1	1
	SSD	15.3	7.8	2.0	Core=88/Freq=2.0/Issue=2/L2=1024k/DS=1280GB/nDS=2/DS_BW=4.5GBs/Net_BW=0.001GBs	5	5	42	1
	DRAM	113.1	78.1	1.4	Core=88/Freq=2.0/Issue=2/L2=1024k/DS=16GB/nDS=16/DS_BW=1.6GBs/Net_BW=0.001GBs	47	52	1198	1
	PCM	1093	337	3.2	Core=128/Freq=0.5/Issue=2/L2=1024k/DS=25GB/nDS=1/DS_BW=32.0GBs/Net_BW=0.001GBs	479	192	15329	1
	MRT	368	112	3.3	Core=128/Freq=0.5/Issue=2/L2=1024k/DS=75GB/nDS=1/DS_BW=32.0GBs/Net_BW=0.001GBs	160	64	5110	1
Recom	Base	1.0	1.0	1.0	Core=56/Freq=2.0/Issue=4/L2=1024k/DS=16GB/nDS=16/DS_BW=1.6GBs/Net_BW=0.500GBs	1	1	1	1
	DRAM	3.0	1.7	1.7	Core=112/Freq=2.0/Issue=2/L2=1024k/DS=16GB/nDS=16/DS_BW=1.6GBs/Net_BW=1.250GBs	1.0	1.0	1.0	2.5
	PCM	6.2	3.4	1.8	Core=128/Freq=2.0/Issue=2/L2=1024k/DS=25GB/nDS=1/DS_BW=32.0GBs/Net_BW=0.500GBs	10.2	11.7	12.8	10.2
	MRT	5.5	2.4	2.3	Core=120/Freq=0.5/Issue=4/L2=1024k/DS=75GB/nDS=1/DS_BW=32.0GBs/Net_BW=1.250GBs	3.4	1.8	4.3	8.5
Search	Base	1.0	1.0	1.0	Core=80/Freq=2.0/Issue=4/L2=1024k/DS=16GB/nDS=16/DS_BW=1.6GBs/Net_BW=0.001GBs	1	1	1	1
	DRAM	2.4	1.4	1.7	Core=128/Freq=2.0/Issue=2/L2=512k/DS=16GB/nDS=16/DS_BW=1.6GBs/Net_BW=0.001GBs	1.0	0.8	1.0	1.0
	PCM	25.4	6.2	4.1	Core=128/Freq=0.5/Issue=2/L2=512k/DS=25GB/nDS=1/DS_BW=32.0GBs/Net_BW=0.001GBs	10.2	2.0	12.8	10.2
	MRT	8.5	2.1	4.1	Core=128/Freq=0.5/Issue=2/L2=512k/DS=75GB/nDS=1/DS_BW=32.0GBs/Net_BW=0.001GBs	3.4	0.7	4.3	3.4

Table 5: Configurations and scale multipliers of the baseline and SSD/DRAM/nanostores designs

Table 5 presents additional data to provide further insight into these results (we focus on the non-shaded rows in this section). Columns 3, 4, and 5 provide the factor of improvement in the energy delay product, performance, and energy efficiency respectively. Column 6 summarizes the attributes of the best configuration chosen by the optimizer, and the last four columns present statistics on the multipliers of improvement in various system attributes. Specifically, *Node*, *OPS*, *DS*, and *Net* refer to the factor of increase in the number of processor sockets, and the total provisioned raw compute bandwidth, datastore bandwidth, and network bandwidth respectively.

The results in Table 5 show that the greatest improvement in resources occurs for the data store bandwidth, resulting from the combination of both the higher per-nanostore 3D-stacked bandwidth and lower per-device capacity. For example, with more than 5000 times higher bandwidth, the three I/O-intensive benchmarks no longer have any data store access bottleneck. With co-located compute, nanostores also allow significantly higher compute and network bandwidths to match the increased data store bandwidth, regaining the balance across system resources to improve performance. By breaking the bandwidth wall in the conventional architecture, processor power density and network aggregate bandwidth now become the new, important system design constraints. (This also illustrates the reason why we capped these variables for a fair comparison with the baseline; Section 4.6 considers relaxing these constraints further.)

Focusing on performance improvement, the biggest benefits stem from the increased parallelism of the nanostore solution that allows greater amounts of compute and network to be provisioned for smaller slices of data capacity. Using *cksum* as an illustrative example, the best PCM-based nanostore design (as determined by the design space search) uses 128 simple cores running at 500MHz in conjunction with the 25GB datastore. The design uses the lowest network bandwidth because this benchmark does not generate network traffic. The raw compute bandwidth increases by a factor of more than 1000 to match the 15000X increase in data store bandwidth, and these two resource improvement together provides three orders of magnitude better performance. (The *OPS* multiplier is lower than the *Node* multiplier due to processor power density constraint, that bounds the performance improvement)

Although *cksum* is an extreme example because of its low processing complexity, *video* illustrates the same performance benefits from higher data store bandwidth and matching, co-located, compute capability. To address the higher compute requirement over *cksum*, the optimal baseline node for *video* now has more cores and less storage capacity. This leads to a lower *Node* multiplier and a lower *OPS* multiplier due to the power density constraint, explaining the lower performance improvement relative to *cksum*.

Sort is a benchmark with balanced compute, data access and network bandwidth requirements. In this case, the network bandwidth becomes the new bottleneck once the datastore bandwidth bottleneck is addressed, limiting the *Net* multiplier and subsequently, the performance improvement. For PCM-based nanostores, only 22 cores are required to match the limited network bandwidth; the Memristor-based nanostore has higher per-device capacity and correspondingly higher *per-node* network bandwidth for a given aggregate network bandwidth constraint, explaining its higher performance improvement.

For the two in-memory benchmarks, a similar analysis can be applied. The relatively smaller performance improvements compared to the IO-intensive benchmarks stem from a smaller *DS* multiplier over the high-bandwidth DRAM interface, and subsequently lower *OPS* and *Net* multipliers due to the power density and network bandwidth caps.

Our detailed analysis also identified (surprisingly) that the significant bandwidth improvements enabled by the 3D-stacked architecture were not being fully leveraged. One reason, as discussed above, is that the constraints on power density and network bandwidth can affect the effectiveness of how well the bandwidth is used. Indeed, our additional experiments (data not shown in table for space) show significantly higher performance improvements when these constraints are relaxed. Furthermore, as discussed in Section 3, our performance model and the COTSon-generated per-core memory bandwidth numbers used as input to the model are both conservative about the effect of improved memory bandwidth on performance, likely contributing further to these conservative results.

Finally, the nanostore design's memory-like datastore latency has huge performance potential for workloads that are random-access dominant and latency sensitive. However, the benchmarks we study are throughput-

oriented and our performance model is mainly bandwidth based; therefore our results do not demonstrate this potential benefit.

4.3 Analysis of energy efficiency benefits

Besides significant performance gains, nanostores also achieve 2X to 16X improvement in energy efficiency. Below we discuss the three main contributors to nanostore's better performance-per-watt.

First, the NVRAM-based data stores are significantly more efficient than the hard drives or DRAM modules used in the baselines due to their better proportionality (no idle power) and lower access energy (technology and 3D-stacking). For I/O heavy benchmarks, the hard drive access energy can be orders of magnitude higher than nanostore; while for in-memory workloads, the large DRAM capacity adds large idle power.

Second, compute co-location with lower per-nanostore capacity leads to the use of low-power, more energy-efficient processor cores. As shown in Table 5, nanostores often choose lower frequency, simpler cores, that are much more energy-efficient due to voltage and frequency scaling. This effect is less dominant for the in-memory benchmarks as they use more powerful cores to satisfy compute demand or low latency.

Finally, having a single-level data store also provides the opportunity to avoid data movement between the logically separate segments of memory and persistent storage. Reducing the number of copy operations can improve performance due to reduced traffic and less energy for the same task, both leading to better energy efficiency. These effects are hard to isolate with the integrated model we consider, but we performed separate experiments to study the elimination of redundant file load and save operations. Our results show significant traffic reduction in *cksum* and *sort*. This translates to about 10% efficiency improvement; the relatively low improvement stems from the efficient data stores in our designs. The benefits are more pronounced when the data access bandwidth is limited. For example, *cksum* gets more than 30% better EDP when per-nanostore bandwidth is 6.4GB/s.

The memristor-based design achieves higher energy efficiency by virtue of having more energy-efficient data accesses, but as discussed earlier, has lower performance compared to the PCM-based nanostore because its higher capacity leads to a smaller *Node* multiplier.

4.4 Applicability of nanostore techniques in other system architectures

As discussed so far, the nanostore design achieves its benefits from a combination of several inter-related factors relating to high bandwidth per gigabyte, matching compute/network bandwidths, and co-location. Of these factors, the nanostore design's improvements to the data access bandwidth are fairly unique in comparison with traditional system architectures. First, cost-sensitive hard drives usually have a floor price (e.g., $30 for a mobile 2.5-inch drive) to amortize non-media costs. This effectively determines the hard drive's minimum capacity, and sets it to a level much larger than for a single nanostore socket. Combined with the inherent low bandwidth of disks, it is not easy to apply nanostore's compute/storage co-location principle

that needs small storage chunks. Second, SSDs using current NAND flash technologies also have limited pins per package (typically shared by 4 dies), having a higher, yet still limited, bound on bandwidth per device. Finally, DRAM and PCIe based SSDs can have much higher bandwidth by exploiting device-level parallelism; however, compared to nanostores, they are still limited by the relatively narrow channels between the compute and the datastore.

To better illustrate the benefits from the nanostore design and the applicability of individual techniques to traditional architectures, Table 5 presents some additional data (shaded rows) listing the best SSD and DRAM-based systems. As before these points represent the outcome of the design space search optimized for the balanced design with the best energy delay product.

The results show that while the new DRAM and SSD-based designs show benefits, the nanostore designs still achieve higher performance at better energy efficiencies. The SSD designs share some of the energy efficiency advantages of lower idle power for the data store but suffer from lower bandwidth per GB; consequently, they have lower performance but higher energy-efficiency relative to the DRAM designs. Note that the best SSD-based designs also choose more efficient processor cores, as suggested by prior work using low-power processors with NAND flash [5][8], but their scaling-down of the processors is much less aggressive than that of the nanostore designs. Interestingly, DRAM-based solutions often pick more powerful cores even if they are allowed to use more efficient, low-power cores. This is because the optimizer chooses to use faster, more powerful cores and hence more processor-power dominating solutions to offset the energy non-proportionality caused by DRAM idle power. In other words, more efficient and higher performance data stores can motivate the selection of more energy efficient processor cores, leading to additive efficiency benefits.

4.5 Other objective functions

Table 6 summarizes the results for the PCM-based nanostore when the optimizer uses other objective functions, under the same thermal and network constraints. For each benchmark, we normalize the performance and efficiency numbers over the same configuration (EDP-optimized baseline). The optimizer clearly chooses different configurations to reach different objectives. EDP is a good objective because EDP-optimized solutions usually have close-to-optimal performance and EE results across the entire table.

Result	Performance			EE		
Objective	EDP	EE	Perf	EDP	EE	Perf
Sort	17.3	11.3	23.9	7.2	8.5	0.9
Cksum	3234.8	674.6	3706.6	15.9	22.1	12.0
Video	337.4	72.9	362.2	3.2	4.8	2.2
Recom	3.4	0.9	3.5	1.8	2.6	1.2
Search	6.2	2.8	8.1	4.1	5.0	2.7

Table 6: Impact of the optimizer's objective function

4.6 Impact of relaxed power density and network constraints

Table 7 visualizes the effect of relaxing the socket power density (32, 50, and 100 W/ cm^2) and network constraints (X1, X4, X16 aggregate network bandwidth) for our benchmarks. All results are normalized to the PCM nanostore design from Section 4.1 (X1 network bandwidth and 32W/cm^2). Darker shades illustrate improved benefits. Allowing higher power density has a positive performance effect for all workloads,

matching our analysis in Section 4.2. Raising the network bandwidth cap only affects the two network-heavy benchmarks (*sort* and *recom*), especially *sort* where the network is the first bottleneck for performance scaling. Power density is the first bottleneck for *recom*, which has to trade core count with higher network bandwidth within the power envelope to get better performance.

Result	Performance									EE								
Watt/cm2	32			50			100			32			50			100		
Net_BW	x1	x4	x16	x1	x4	x16	x1	x4	x16	x1	x4	x16	x1	x4	x16	x1	x4	x16
Sort	1	4	22	1	4	22	1	4	22	1.0	1.0	0.8	1.0	1.0	0.8	1.0	1.0	0.8
Cksum	1	1	1	2	2	2	2	2	2	1.0	1.0	1.0	0.8	0.8	0.8	0.8	0.8	0.8
Video	1	1	1	2	2	2	3	3	3	1.0	1.0	1.0	0.7	0.7	0.7	0.6	0.6	0.6
Recom	1	2	2	1	3	3	1	3	6	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9
Search	1	1	1	2	2	2	2	2	2	1.0	1.0	1.0	0.6	0.6	0.6	0.6	0.6	0.6

Table 7: Impact of thermal/network constraints

4.7 Discussion

Endurance is an important issue to consider. For the peak memory bandwidth we consider, in theory, storage wear out can occur in 2 years for PCM or 11 years for Memristor based on nanostore capacity and endurance. However, in practice, not all applications sustain rates at that level and the average across the application is much lower, leading to much longer lifetimes across the array. Wear-leveling schemes must still be used to spread writes across the entire memory to prevent early failure of hot data blocks. Assuming a previously proposed approach – start-gap wear leveling – at an efficiency of 90% of optimal wear-leveling (shown to be realistic for OLTP/database workloads [26]), and using the memory write bandwidths from our simulations, we estimate per-socket lifetimes of 7-18 years for our benchmarks on the PCM-based design. Nevertheless, techniques that carefully manage wear-out warrant further study.

Another important issue is around scaling of workloads. The performance improvements from nanostores stem from the larger distributed scale of the workloads, with scaling factors ranging from 100 to 500. Even with the workloads we consider that are targeted at large-scale distributed implementations, such scaling is likely to pose challenges. Our idealistic assumptions around scaling are not meant to gloss over the challenges of scaling, but rather to provide an upper bound on the potential benefits. However, it is worth noting that over the decade from 1998 to 2009, Google's infrastructure is reported to have scaled performance (queries processed/day) by 1000X while scaling the infrastructure by 1000X [24].

In this paper, we focus primarily on architectural and technology implications for best future designs, but cost is another issue that also needs to be considered. Current flash memories are about an order of magnitude higher cost on a $/byte basis compared to disk. The NVRAM memories we consider in this paper have the potential to lower these costs by more aggressive stacking and simpler fabrication processes. The improved energy efficiency of our design can also further lower total costs of ownership. Based on these observations, we expect the nanostore design to be competitive in costs compared to traditional designs, but this needs to be validated with further study. We are working with vendors to determine costs projections for PCM/Memristor technologies and expect to have more discussion on costs in the final paper.

5. RELATED WORK

Several of the principles leveraged in the nanostore design have been studied in prior work. The iRAM and PIM work [13] [25] examine integrating a processor with DRAM in the same process, and consider benefits with vector streaming programming models. While thematically similar, our implementation is significantly different in several ways, including the use of lower-power commodity processors with a disk-less flat hierarchy based on non-volatile memory, and a distributed software model. Active Storage [30] incorporates compute closer to disk, but only in the form of more powerful disk controllers for offloading and streaming. The main processor is still a deep hierarchy away.

Recently, the RAMCloud project [11] has proposed distributed systems where all data resides in DRAM. Their planned research primarily focuses on the software stack, around low-latency RPC, durability, data model, scaling, and consistency. While several of their motivating arguments are similar to ours, we differ in our assumptions around all data residing in 3D-stacked non-volatile RAM and in our architectural explorations around balanced system designs. Our results validate their projected performance but show even better benefits from our approach. Nevertheless, many of the software ideas are likely applicable to our design as well.

Other recent studies have examined using lower-power (thin or wimpy) cores for energy efficiency [5] [9] [21] [31] while also being aware of the impact on quality of service [29]. There has also been prior work on ultra-low-voltage core design [37]. Like these studies, we also explore the benefits from better balanced designs, but synergistically in combination with rethinking compute-data proximity and hierarchy.

Recent architectural proposals have studied 3D stacking and demonstrated its viability, and its benefits for improved bandwidth and memory redesign (e.g. [12] [38]). These studies do not consider large-scale distributed systems or co-location with non-volatile memory.

Previous studies have discussed different ways to use existing and emerging non-volatile memories e.g. Flash [8] [9] [16] [17], PCM [18] [27] [38] [39], and Memristors [28] [32] [33]. While Flash memories have been shown to be effective as storage, disk replacement, or disk cache [9] [1] [16] [17], their latency and endurance limitations make them inapplicable for our work. Recent work on PCM has examined its use both as Flash replacement and as memory replacement (including in 3D-stacked configurations [38]), but there have been no studies that have focused on simplification of the data hierarchy, in the context of data-centric workloads. Prior studies have prototyped and evaluated Memristors [32] [33], but we are not aware of any architectural studies using Memristors.

Several studies have proposed optimizations to improve endurance [18] [27] [38] [39] and others have identified potential improvements in the future [4]. Other studies have examined special-purpose architectures optimized for specific data-centric workloads including use of GPUs [23], FPGAs [2], and even ASICs [6] and co-design of hardware and software for data-centric workloads. (e.g., MonetDB [7], MapReduce [10]). Such optimizations would be applicable in our solution as well.

6. CONCLUSIONS

With data and data-centric applications on the rise (steeply), there is an emerging market for new system designs targeted at these workloads. At the same time, emerging technologies like 3D-stacked non-volatile memories are likely to disrupt traditional assumptions around storage latency and bandwidth. In this paper, we argue that the best (and most intuitive) way to leverage the confluence of these application and technology trends is a radical approach that co-locates processors with non-volatile storage eliminating many intervening levels of the storage hierarchy. Our primary contributions are in developing the design of such an architecture and evaluating its potential benefits and implications.

Specifically, we present nanostores, a new building block for data-centric system design. A nanostore is a single-chip computer that includes 3D-stacked layers of dense silicon non-volatile memory with a layer of compute cores and a network interface. A large number of individual nanostores communicate over a simple interconnect and run a data-parallel execution environment like MapReduce to support large-scale distributed data-centric workloads. The key aspects of our approach are large-scale distributed parallelism and balanced energy-efficient compute in close proximity to the data. Together, these allow nanostores to potentially achieve significantly higher performance at lower energy.

Using an evaluation model and a benchmark suite that we newly designed for this study, we demonstrate orders of magnitude improvements in performance at significantly better energy efficiency for key classes of data-centric workloads. We also point out key challenges that need to be addressed to leverage this potential including scalable software design and improved network subsystem design.

While our results are promising, we believe we have only scratched the surface of what is possible. We are currently examining the rich architectural space enabled by nanostores, including heterogeneous designs and integrated optics. There are also interesting opportunities for software optimizations including new interfaces and management of persistent data stores. Looking further out, the large scale and low latency of our designs will likely enable new previously-not-possible applications for more sophisticated insight generation across larger diverse multiple data sources; the corresponding hardware-software codesign provides rich opportunities for future research.

REFERENCES

[1] Fusionio. In *http://www.fusionio.com*.

[2] Netezza. In *http://www.netezza.com*.

[3] COTSon: Infrastructure for system-level simulation. In *MICRO 41 Tutorial*, 2008.

[4] ITRS roadmap. In *http://www.itrs.net/*, 2009.

[5] David Andersen, Jason Franklin, Michael Kaminsky, Amar Phanishayee, Lawrence Tan, and Vijay Vasudevan. FAWN: A fast array of wimpy nodes. In *SOSP*, 2009.

[6] Shinsuke Azuma, Takao Sakuma, Takashi Nakano, Takaaki Ando, and Kenji Shirai. High-performance sort chip. In *HotChips*, 1999.

[7] Peter A. Boncz, Martin L. Kersten, and Stefan Manegold. Breaking the memory wall in MonetDB. *Commun. ACM*, 51(12):77–85, 2008.

[8] Adrian M. Caulfield, Laura M. Grupp, and Steven Swanson. Gordon: An improved architecture for data-intensive applications. *IEEE Micro*,

30(1):121–130, 2010.

[9] Adrian Cockcroft. Millicomputing: The future in your pocket and your datacenter. In *USENIX Conference, invited talk*, 2008.

[10] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. *OSDI*, 2004.

[11] John Ousterhout et. al. The case for RAMCloud: Scalable high-performance storage entirely in DRAM. In *http://ilpubs.stanford.edu:8090/942/*, 2009.

[12] T. Kgil et al. PicoServer: Using 3D Stacking Technology To Enable A Compact Energy Efficient Chip Multiprocessor. In *ASPLOS*, 2006.

[13] M. Gokhale, B. Holmes, and K. Iobst. Processing in memory: the terasys massively parallel PIM array. *Computer*, 28(4):23–31, Apr 1995.

[14] James Hamilton. Internet-scale service infrastructure efficiency. In *Keynote, ISCA*, 2009.

[15] G.C. Han, J.J. Qiu, L. Wang, W.K. Yeo, and C.C. Wang. Perspectives of read head technology for 10 tb/in recording. In IEEE Trans on Magnetics, volume 46, 2010.

[16] T. Kgil and T. Mudge. FlashCache: a NAND flash memory file cache for low power web servers. In *CASES*, 2006.

[17] Taeho Kgil, David Roberts, and Trevor Mudge. Improving nand flash based disk caches. In *ISCA*, 2008.

[18] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting phase change memory as a scalable dram alternative. In *ISCA*, 2009.

[19] Dean Lewis and Hsien-Hsin Lee. Architectural evaluation of 3D stacked RRAM caches. *IEEE 3D System Integration Conference*, 2009.

[20] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. McPAT: An integrated power, area and timing modeling framework for multicore and manycore architectures. In *MICRO*, 2009.

[21] Kevin Lim, Parthasarathy Ranganathan, Jichuan Chang, Chandrakant Patel, Trevor Mudge, and Steven Reinhardt. Understanding and designing new server architectures for emerging warehouse-computing environments. In *ISCA*, 2008.

[22] Peter Lyman and Hal R. Varian. How much information. In *http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/*, 2003.

[23] Wenjing Ma and Gagan Agrawal. A translation system for enabling data mining applications on gpus. In *ICS*, 2009.

[24] Marissa Mayer. The physics of data. In *Talk at Xerox PARC*, August 2009.

[25] David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberly Keeton, Christoforos Kozyrakis, Randi Thomas, I Thomas, and Katherine Yelick. A case for intelligent RAM: IRAM. *IEEE Micro*, 17, 1997.

[26] Moinuddin K. Qureshi, John Karidis, Michele Franceschini, Vijayalakshmi Srinivasan, Luis Lastras, and Bulent Abali. Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling. In *MICRO 42*, 2009.

[27] Moinuddin K. Qureshi, Vijayalakshmi Srinivasan, and Jude A. Rivers. Scalable high performance main memory system using phase-change memory technology. In *ISCA*, 2009.

[28] T. Raja and S. Mourad. Digital logic implementation in memristor-based crossbars. In *ICCCAS*, 2009.

[29] V. Reddi, B. Lee, T. Chilimbi, and K. Vaid. Web Search Using Small Cores: Quantifying the Price of Efficiency. In ISCA, 2010.

[30] E. Riedel, C. Faloutsos, G.A. Gibson, and D. Nagle. Active disks for large-scale data processing. In *IEEE Computer*, vol 34, , 2001.

[31] S. Rivoire, M. A. Shah, P. Ranganathan, and C. Kozyrakis. JouleSort: a balanced energy-efficiency benchmark. In *SIGMOD*, 2007.

[32] W. Robinett, G.S. Snider, P.J. Kuekes, and R. S. Williams. Computing with a trillion crummy components. In *CACM*, vol. 50, 2007.

[33] D.B. Stukov, G.S. Snider, D.R. Steward, and R.S. Williams. The missing memristor found. In *Nature*, volume 453, pages 80–83, 2008.

[34] Richard Winter. Why are data warehouses growing so fast? In *http://www.b-eye-network.com/view/7188*, 2008.

[35] Zichen Xu, Yi-Cheng Tu, and Xiaorui Wang. Exploring power-performance tradeoffs in database systems. 2010.

[36] Fan Yang, Wen Su, Huibiao Zhu, and Qin Li. Formalizing mapreduce with CSP. 2010.

[37] B. Zhai, R. G. Dreslinski, D. Blaauw, T. Mudge, and D. Sylvester. Energy efficient near-threshold chip multi-processing. In *ISLPED*, 2007.

[38] Wangyuan Zhang and Tao Li. Exploring phase change memory and 3d die-stacking for power/thermal friendly, fast and durable memory architectures. In *PACT*, 2009.

[39] P. Zhou, B. Zhao, J. Yang, Y. Zhang. A durable and energy efficient main memory using phase change memory technology. In *ISCA*, 2009.

[40] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale Parallel Collaborative Filtering for the Netflix Prize. In *Algorithmic Aspects in Information and Management*, 2008.