Empowering Architects and Designers: A Classification of What Functions to Accelerate in Storage

Chen Zou
chenuou@uchicago.edu
University of Chicago

Andrew A. Chien
achien@cs.uchicago.edu
University of Chicago

ABSTRACT
Storage acceleration is a topic of widespread study and innovation. We seek fundamental understanding of 1) what computations to offload to benefit application performance and scalability and 2) how the properties of such computations shape offloading benefits. Using raw-data analysis as an exemplar, we study acceleration benefits using SparkSQL operating in a cloud data center, analyzing where and how speedup and scalability are improved.

From this base, we analyze 14 function offload candidates drawn from 17 research studies to create a classification to help the application or hardware system designer. We analyze from the perspective of feasible acceleration, network data reduction, and software change - for application and system software. These studies identify functions that are clear wins and should be the target of early "computational storage", those that give clear benefit, but face software challenges to support, and finally those that are less compelling.

1 INTRODUCTION
The past two decades have seen explosive growth in large-scale data, ranging from scientific research (particle accelerators, molecular dynamics, weather modeling and so on), internet services (web search, e-commerce, social networking, music and video streaming) to nearly every aspect of the economy and society through the explosion of data science and analytics. More recently, the success of machine learning in delivering function from data has accelerated this trend. These data sets range from a few hundred gigabytes to 100’s of petabytes [4, 9], and their efficient storage and processing is the raison d’etre of massive and rapidly growing global networks of both HPC and commercial data centers [1, 3, 8].

While rotating storage (HDDs) remains the cheapest capacity storage, the continued scaling of NAND flash[14] has reduced the cost of SSD capacity sufficiently such that its superior bandwidth and IOPS performance make it a mainstay of the modern datacenter. Leading enterprise SSDs can reach 3.5 GB/s and 720K IOPS [13], and consumer SSDs exceed 3 GB/s and 300K IOPS [10] at a much lower cost.

How to best divide computation across compute and storage has long been important to system performance and cost-effectiveness, and the subject of research going back to the 1990s [18, 42]. But recent rapid increases in storage device capability have shifted scaling and cost bottlenecks in the modern HPC or cloud datacenter architecture onto the CPU and interconnect (see Figure 1). This changing balance has reopened a set of critical research questions about what kind of acceleration is beneficial, how to redistribute work and rebalance systems, should acceleration be added to compute or storage, etc. The community has seen an explosion of research [22, 24, 32, 39, 45, 46] and innovation [37, 47] exploring specific approaches to storage acceleration. Terms such as "cognitive storage" and proposals for full compute functions in storage nodes consider a reversion to 1st generation datacenter architectures as seen in Figure 1.

In this exciting, but chaotic environment, our approach is to address a focused question. Presuming datacenter operators want to preserve the modern datacenter architecture and associated usage model (allocate compute, share storage), we explore the questions of what computations give benefits when accelerated and offloaded to storage? And, can we provide more general guidance to applications and datacenter hardware architects on what acceleration to include on storage nodes/devices?

We first perform a case study, using a raw data analytics system, exploring the performance impact of storage-side acceleration. We consider both in-storage node and in-storage device acceleration, assessing opportunities for improved performance and system scalability. The study gives insights into the best location for acceleration and its quantitative benefits. Second, we perform a broad literature study of the computations being considered for storage acceleration, analyzing the properties of these computations and how said properties affect performance benefit. This analysis builds on insights from our raw data analytics case study and produces a classification that guides both datacenter hardware architects and application designers in understanding what functions can benefit from storage offload, as well as which require significant software changes – application and storage system – and thus have benefits that are more difficult to realize. Specific contributions of the paper include:

- Using a case-study of raw data analytics with a multi-stage heterogenous pipeline structure, we characterize quantitatively the benefits of storage-side in-node and in-device acceleration. Latency improves 4.5x and scalability 14.9x (geometric mean, TPC-H benchmarks)

- Analysis of key properties (acceleratable, network data reduction) of 14 proposed computations for storage acceleration and used in 17 research studies of proposed acceleration systems. These computations are drawn from filesystem, database, and application computations.

Figure 1: Modern Datacenter Architecture: Separate compute and storage layers.
We study on-demand raw-data online analytical processing (OLAP), which stem from a carefully customized architecture, and traditional hardwired implementation (ASIC vs. FPGA). The study explores acceleration benefits and impacts in storage-side acceleration.

4 METHODOLOGY

4.1 Modeling

Workloads. We use all 22 TPC-H queries [16] on raw data for evaluation. These queries span simple select-project-join structures to more complicated ones with expensive expressions like regex matching and large intermediate attributes. The queries operate on raw data in tabular format with different fields separated by delimiter ‘|’. Each field is in text format rather than binary, exercising the flat parsing and data type conversion dimensions of raw data processing. We decompose the TPC-H queries as below.

Task Elements:

- Storage IO: Read raw data from storage devices.
- Parse: Parse data for column and row boundaries for selection and conversion (i.e. text int/date to binary)
Empowering Architects and Designers: A Classification of What Functions to Accelerate in Storage

Figure 3: Task elements for a raw data analytics application, data flows from the storage, and with offload is processed, then transferred to the CPU for final query logic.

- Select: Extract needed columns from a database table.
- Filter: Filter rows based on a predicate.
- Transfer: Move data from storage to compute node for analysis.
- Query logic: Execute TPC-H query logic on data at the compute node.

These elements are depicted in Figure 3. All three of the parse, select, and filter operations can be run on the CPU or offloaded to the storage-side. Network transfer is shown for one particular mapping, where parse, select and filter are run on the storage node.

4.2 System Architecture

Software. We use SparkSQL[15] as the raw-data analysis system. Its query optimizer, Catalyst, generates predicates and pushes them into the DataSource extension mechanism [11] to reduce data. We map the data reduction process onto storage-side acceleration.

Hardware. Compute nodes are a single Skylake-SP CPU (32 threads), and HDDs (0.1GB/s, SATA3.0) and SSDs (4GB/s per device) are considered on the storage nodes.

Compute and storage nodes are connected by 40GigE or NVMe over Infiniband. We consider two storage-side execution units: First, a data transformation accelerator (the UDP [25, 26] as discussed in Section 2.2) whose low power and small size make it suitable for integration in-device or in-node. The UDP delivers high performance on data transformation, scanning, and pattern recognition. Second, we consider an ARM Cortex A53 (4 cores), an embedded processor design, often included in high-end SSDs and with comparable area and power to the UDP. A single core of the embedded processor is 16x slower (SPECint2000/2006) than a single Skylake-SP core.

Data transfer between compute and storage nodes can be limited by PCIe connections or the datacenter network. We consider the interconnect throughput to be sufficient as modeled by 4GB/s (4x PCIe 3.0) for each SSD.

Hybrid Simulation. Application performance is computed by a hybrid simulation technique that measures task elements on a 32-core CPU, storage, acceleration, and network separately and scales them appropriately for the modeled system. Three components of the total execution time of SparkSQL raw data analytics are all captured in the execution logs. Query logic execution time is unmodified, accelerated tasks are scaled based on the accelerator performance model (for UDP software simulator and cycle time from hardware design and for ARM by relative performance), and StorageIO time is scaled considering the modeled storage system bandwidth.

4.3 Metrics

We report per task-element runtime in seconds for each query. In sum, these are the end-to-end application latency. Unless otherwise specified, the runtime corresponds to a 32-thread implementation on one CPU. We also report CPU-scalability (the number of SSDs that can be driven by one CPU at the same utilization as in our Baseline system (1 CPU, 4 SSD).

5 CASE STUDY: RAW DATA ANALYTICS

First, we present studies that explore the performance implications of offloading of several types and in several locations (in storage node, in storage device, compute-side). We consider different mappings of raw data analysis task elements to CPU and acceleration. The results illustrate the complex performance dynamics between application stages, mapping, and resources, illuminating a variety of design issues for storage acceleration. Second, we consider how each configuration affects CPU-scalability, a critical cost-effectiveness issue. Finally, we consider the cost of acceleration hardware, presenting performance normalized for silicon area and power.
5.1 Scaling Disk Bandwidth

We start with a single thread of a compute node CPU and a single hard disk drive (HDD) (as in Section 4.1). To show system balance, we increase IO bandwidth by 160-fold from the HDD configuration to a 4-SSD configuration (16 GB/s bandwidth). The result, shown in Figure 4, illustrates the effects of growing storage system performance. The increase in IO bandwidth from the HDD configuration (lighter, on left) to the 4 SSD configuration (darker, on right) reduces total runtime only slightly. Since query logic, parse, filter time (all on CPU) dominate the runtime, this and further increases in IO bandwidth motivate storage acceleration.

With multicore chips, more threads can be applied to the compute bottleneck. To model this, we scale compute node runtimes for 32 threads (assuming perfect speedup). This produces the results in Figure 5 which we call the Baseline (one CPU, 4 SSD). Despite performance overestimation with perfect speedup, query logic, parse, and filter tasks (on CPU) still dominate the runtime while IO remains just below 10%, a small percentage of the overall runtime.

5.2 Storage-side Offload

Next, we consider how storage-side offload affects performance. First, we consider a cheap storage node augmentation, adding an embedded-CPU, a Cortex A53 (4 cores) to execute offloaded tasks. This configuration may be similar to Amazon S3 Select storage nodes [2], rumored to use ARM cores for column select and row filtering tasks.1 Surprisingly, this offloading reduces performance dramatically, increasing runtimes 100-fold! (see Figure 6). There are two reasons for this. First, the much slower ARM cores cause the runtime for parsing and filtering to increase 120-fold. These tasks accounted for 50-85% of runtime on the CPU, and when shifted to the A53, their share increases to more than 99% (see Figure 7). In this case, offload to the A53 shifts the bottleneck and aggravates it. So, if we do not have suitable hardware accelerators, offloading can be harmful to performance. Second, the data size reduction of storage-side filtering alone yields little benefit as the data transfer is a negligible fraction of overall runtime. From these results, it is clear that offloading from the compute nodes does not always increase application performance.

5.3 Storage-side Acceleration

As we have seen, offloading can reduce the compute-node CPU load, but if the task is slowed by offloading, there may be little benefit – or even performance degradation. We next consider accelerators that perform important storage tasks faster and at lower power and cost than the compute node CPU. For the next experiment, there is a UDP accelerator ([25, 26] and see Section 2) in each storage node. With the acceleration of the offloaded parse and filter tasks, the performance improvement compared to Baseline is significant, ranging from 1.7x to 3.9x, and with an average of nearly 2.5x (see Figure 8). With accelerated parse and filter, the contributions of Storage IO, Parse & Filter, and Query logic are comparable – none dominates. This storage-side accelerated system is 200x faster than the naive offload to A53 results shown in Figure 6.
Empowering Architects and Designers: A Classification of What Functions to Accelerate in Storage

5.4 In-Storage Device Acceleration

We next consider acceleration in each storage device (one UDP per SSD). This increases hardware expense for storage, and may be possible for small accelerators such as UDP, but infeasible with less efficient approaches such as FPGA’s [32, 43]. The obvious advantage of in-storage device acceleration is that acceleration capability scales with storage bandwidth.

As shown in Figure 9, the increase in acceleration bandwidth delivers a further increase performance of nearly 50%. Specifically, performance increases by 1.9x to 6.3x with an average of 3.8x compared to Baseline. Clearly, more acceleration is desirable if it can be afforded. The per-task breakdown shows that the parse and filter bottleneck is eliminated with in-storage device acceleration, leaving the system largely limited by Storage IO and Query Logic on the CPU. We analyze these benefits from another perspective, evaluating net system scalability in Figure 10.

5.5 CPU-scalability

The storage accelerated system shifts computation to the storage nodes, increasing the number of SSDs that a CPU can drive. The CPU scalability metric quantifies this change in system balance as shown in Figure 10. For the average performance over the TPC-H benchmarks, we can achieve CPU scalability increase for UDP-accelerated systems to nearly 64 SSDs, an increase of 14.9x over the Baseline. The CPU-scalability increase for a best single TPC-H query is even higher, to 139 SSDs, an increase of 24.8x.

5.6 Normalized Performance

Acceleration’s performance benefits have a cost – increased silicon area and power. To consider these increased costs, we compute normalized performance from the geometric mean of runtimes for the TPC-H benchmarks. First, we normalize for silicon area which is estimated from die shots [5, 6]. The results show that performance efficiency does not improve for the A53 and Skylake accelerated cases because no speedup is achieved (Figure 11). However, the storage-side and in-device acceleration is highly efficient, and net 3x and 4x improvements in normalized performance respectively. The power efficiency analysis yields similar results. So in this case, the acceleration benefits are much greater than the costs.

5.7 Summary

Our studies show that offloading for raw data analytics is tricky. Offloading does not always increase performance, but in a number of cases, significant performance increases can be achieved in cost-effective fashion. With these insights, we undertake a greater challenge, understanding in general when it is profitable to offload computation to storage for system performance and scalability.

6 THE RESEARCH LANDSCAPE OF STORAGE ACCELERATION

With inspiration from the many studies, including ours (Section 5), and the expansive interest in storage acceleration, we consider more general questions –

(1) what computations give benefits when accelerated and offloaded to storage? (application guidance)

(2) what general guidance is there on what acceleration to include on storage nodes/devices? (cloud architect guidance)

Part of this situation is depicted in Figure 12 where an application designer considers whether to deploy elements of an application across compute and storage nodes of the system. On the other hand, the question for the cloud data center architect (or storage system designer) is what acceleration to add to storage and compute nodes for maximum benefit and minimum cost.

In short, the criteria is “is storage acceleration the most efficient approach for the system – hardware architecture and applications”, not simply “does storage acceleration increase performance”. There are numerous functions that increase performance if accelerated in storage, but not in a fashion most efficient for the system.

In this study, we consider these questions for a vertical slice, CPU - memory - network - storage node - SSDs, as shown in Figure 12. We defer consideration of more complicated NxN and NxM scenarios and storage sharing for future studies.

6.1 The Spectrum of Functions for Offload

To identify candidates for offload, we surveyed the research literature for file systems, databases, and some miscellaneous application domains [12, 19, 20–28, 32, 34, 38, 39, 41, 43, 45, 47–49]. In Table 1, functions (candidates for offload) are shown as columns and the numerous systems that explore storage acceleration are shown as rows. The table illustrates the breadth and diversity of potential storage acceleration, and collectively, they are the forefront of systems research for storage acceleration.
For filesystems, the focus has been on block-oriented operations to transform data – for security, reliability, and density. The block-oriented focus arises naturally from the block-devices found in storage systems and the studied operations can be 1-to-1 (i.e., cryptography or compress), many-to-1 (deduplication), 1-to-many (replicate, erasure code) with the corresponding data IO and network transmission requirements.

Databases operate on tables of tuples or columns of attributes. As such, the operations explored are all abstract database operations such as filter, select, summarize, and "parse" operations for unstructured data items (often stored in strings). While these databases operate logically on individual tuples or values, they can be collected into large groups or block operations. The work per item varies from small to large for filter and summarize with variation arising from parsing, pattern matching, and operator complexity. Select on structured data is typically fixed low cost. A common exploitation of database operators that provide data reduction is predicate pushdown on the CPU. This query plan optimization can produce benefits independent of acceleration and offload.

Application examples come directly from expensive computing or data reshuffling operations in "big data" applications. Easy to write in conventional programs, operating in main memory, storage acceleration shifts these common computations such as transpose, neural networks, or statistical modeling to accelerator hardware in the storage such that they could even operate autonomously.

Despite numerous studies, overall coverage of acceleration opportunities is sparse – we are just at the beginning. Our goal is to generalize from the many point studies – each reporting performance benefits for a specific acceleration approach. Such studies often fail to normalize for acceleration costs (high for FPGA’s and full CPU’s), or consider compute-side acceleration as an alternative. We focus on separating the benefits of adding acceleration and the benefits of its location. In short, our criteria is "does acceleration help?" and "is the most efficient place for the acceleration in the storage system?" Numerous functions that increase performance if accelerated at storage give greater benefit if placed elsewhere.

### 6.2 Analyzing Acceleration and Data Reduction

We consider three groups of acceleration candidates drawn from Table 1, based on their source – file systems, data analytics / database, and applications. For each group, we classify the functions based on two dimensions –

1. Is the function acceleratable?
2. Does offloading reduce network data transmission cost?
These dimension capture first the traditional notion of beneficial computation hardware acceleration, where the accelerated implementation provides latency, throughput, or energy benefits. The second captures the notion that acceleration specifically on the storage improves system performance and scalability by increasing the exploitation of plentiful storage bandwidth. We consider not only the total data transmitted, but in some cases also the nodes amongst which it must be communicated – as that can affect network cost.

In Figure 13 we classify the functions from our survey based on our two criteria - acceleratable and network data reduction. For filesystem functions, cryptography, compress and deduplicate are all acceleratable. However, offloading them to storage does not reduce data transfers between compute and storage. Cryptography generally does not reduce size at all. And compress and dedup if done at the CPU, produce reduced-size data to be transmitted to storage. If compress and dedup are offloaded, full-sized data must be transmitted - offloading increases the data required to be transferred. Erasure coding, serialize/archive, WAL replay, and replicate all involve multiple storage nodes, creating a more complicated situation. In these cases, offloading can segregate required data transfer within the storage nodes (cluster), effectively reducing data traffic between compute and storage; it can also be contained to a special storage network.

Further in Figure 13 we see that for database functions, offloading studies have focused on the operators that reduce data size - and hence network traffic. For filter the computation per tuple can vary widely (from arithmetic comparisons to complex regular expression matching). Summaries can range from max to complex computation operators, creating acceleration opportunities. Further, if the operators have low selectivity, these functions can reduce data transfer cost as well as follow-on computation.

Finally, we consider application kernels in Figure 13, including compute and shuffle intensive kernels such as statistical modeling (KNN classification or feature selection), neural network inference, and transpose. These applications all depend heavily on structured data movement and computation, and are thus amenable to acceleration through parallel hardware such as vector and matrix multiplications operations. If the kernels can be offloaded entirely to storage and confined to a single node, data transfer costs can be reduced, by avoiding the transfer to the CPU completely.

Examples of these challenges are varied. We discuss several examples from Figure 14, where they are marked in red. For example, offloading an application program for neural-net inference to storage requires mapping between application, accelerator, and perhaps storage address space. If the input array is a subset of a multi-dimensional or sparse array, the mapping may be complicated, and other problems ensue with microcontrollers with smaller addressing ranges. Traditional solutions (shared address space, virtual memory, or coherent memory) all increase hardware complexity and create variability in performance. A second example is WAL replay offload as in Amazon’s Aurora database; which shifts the log replay into to the storage layer. Storage-offloading WAL replay requires running a collection of processes distributed across storage nodes. The nodes must be quite capable - able to initiate messages and IO operations in order to persist changes from the log in storage. These capable nodes must support variable computational requirements and high resource requirements [47]. A third offload type, functions such as compress or cryptography are fixed cost, local-data only operations with well-defined semantics. Acceleration hardware for them can be designed and provisioned specifically matched to performance needs, and with minimal software changes. Beyond examples such as WAL replay, all of the application-defined offload candidates will require special scheduling support from the system software. Because the application changes the data size, work, or even the function, their requirements are not known in advance, and can also be variable. This is a problem in shared storage resource applications. Other computations that present difficult scheduling issues include parse, statistical modelling, serialize/archive (red in Figure 14).

6.4 Example Offload Candidates

We analyze exemplary offload candidates with respect to our three criteria - acceleratable, network data reduction, and software change.

We start with parse, drawn from databases. Costs per tuple for parsing can be expensive, signaling significant acceleration opportunities. And its branch-intensive properties fit well with accelerators like UDP [26]. Parsing also brings moderate data reduction benefits during type conversion (e.g. 10 byte text int -> 4 byte int). However, parse costs can be highly-variable subject to the complexity/structure of the item being parsed creating scheduling challenges for system software in the storage system. As a result, parsing is (acceleratable, data reducing, need software change). Its performance benefits encourage storage-offloading, the software challenge must be addressed.

Next consider compress, an important file system function. Compress generally operates on blocks, is amenable to acceleration, and easy software integration on blocks. Compress reduces the data size, but in a system does not provide network data reduction benefits when not offloaded (reduced-size data sent from compute to storage). Offloaded compress gives zero network data reduction benefits. Thus, compress is an (acceleratable, no data reduction, no software change) function.

Another distinctive example is write-ahead log replay. WAL replay was originally designed to achieve crash consistency, replaying database changes during recovery. Amazon Aurora [47] offloads WAL replay to the storage layer as a replacement for dirty page

6.3 Analyzing the Need for Software Change

While most storage-side acceleration research focuses on speeding computation, such acceleration can create new software challenges. First, application software generally requires access to the full virtual memory of the application process (address space). Shifting an application computation to a storage-side accelerator creates potential addressing and locality challenges. Second, offloaded application-defined functions can have variable requirement (runtimes), creating a new resource management/scheduling problem for system software on the storage system - a critical shared resource. Error handling and failures further complicate the situation. We highlight (red) functions that create such software change challenges in Figure 14.
Figure 13: Functions plotted - Acceleratable (y-axis) versus Network Data Reduction (x-axis)

Figure 14: Application rewrite and system resource management - software challenges for offload

flushing during normal operation of the DBMS. This refactored software architecture reduces the network data transmission from the database frontend cluster to the storage cluster, sending just modified tuples instead of whole dirty pages. Further WAL replay update traffic is confined to the storage system. Because the log replay operations have complex consistency and reliability requirements, acceleration is difficult. So consider WAL replay as a (not acceleratable, data reducing, heavy software change) function.

6.5 A Proposed Classification Framework

Distilling the discussions above, we produce a classification framework for application designers and system architects. Directly, the framework seeks to answer the question: what functions should be offloaded to storage?

The framework uses three attributes: acceleratable, data reduction, need software change. Acceleratable reflects whether the function can be accelerated with specialized hardware. Data reduction captures whether the function reduces network data traffic - by selective transmission, recoding, or perhaps changing receivers. Finally, need software change indicates whether software rewrite or new functionality is needed. Framework insights:

Hardware Architects. should design storage acceleration hardware to support all functions classified as (acceleratable, data reduction, no software change). These functions will provide both critical path reductions as well as scalability, enabling exploitation of the growing NAND Flash/SSD bandwidth. Furthermore, these functions do not require software change, to they will find rapid adoption. Beyond these attributes, some (acceleratable, data reducing, need software changes) functions are also worth considering, if software pioneers can be found.

The (acceleratable, no data reduction, *) functions are marginal. Offloading acceleration in storage may not give the best benefit. For example, as in Section 7 offloading compress decreases system performance; compute-side acceleration is a better choice.

Application Designers. should consider (acceleratable, data reduction, *) functions for storage-side offloading. Offloading these functions will provide critical path reduction and improved scalability. Those that require no software change are easiest, but as an application designer addressing those that need software change may also be accessible.

The (acceleratable, no data reduction, *) functions can be exploited opportunistically if acceleration hardware is available. Here compute-side acceleration may well be more beneficial than storage-side, and therefore preferable.

Beyond these initial insights for hardware designers and application designers, the framework also creates a clear roadmap, and ordered progression of key challenges to solve to enables the next best candidates for storage offload. The next step is the solve the software challenges for the functions in (acceleratable, network reduction, need software change) and then (not acceleratable, network reduction, need software change).

7 EXPLORING THE CLASSIFICATION

We present performance measurements for a variety of offloading function examples. These results substantiate hardware architect and application designer recommendations and thereby demonstrate the power of our classification framework.

7.1 Metrics

We consider three metrics for performance: 1) Runtime (seconds, end-to-end application latency), 2) Network Traffic (gigabytes transferred), and 3) CPU-scalability (the number of SSDs to reduce Storage IO time to 20% of total runtime). Experiments assume two storage side execution unit configurations: either a 4-core x86 Skylake CPU as the storage-side general-purpose execution unit (denoted as ‘storage offload’), or an application-specific accelerator (denoted as ‘storage accel’). The Skylake CPU is more powerful than realistic storage-side CPUs, as power and cost limits are significant. We will see acceleration is needed to make offloading effective.

7.2 Does ‘Acceleratable’ and ‘Data Reduction’ Justify Storage Offload

We evaluated exemplars of (acceleratable, data reduction, *) functions – filter, parse, and neural network to see if storage offload is the best system configuration (highest efficiency).
Empowering Architects and Designers: A Classification of What Functions to Accelerate in Storage

Figure 15: Filter - acceleratable, data reduction, no software change

Figure 16: Parse - acceleratable, data reduction, software change

Figure 17: Neural network inference - acceleratable, data reduction, software change

Figure 18: select - not acceleratable, data reduction, software change

(acceleratable, data reduction, no software change) - Filter. Figure 15 presents runtime (latency by elements), network traffic and SSD scalability for filtering on parsed TPC-H Q6 ‘lineitem’ table of 10GB. The results show that offloading the filter to storage-side acceleration delivers improved performance. Storage offloading eases the network bottleneck by sending only the needed data to the CPU, as shown by the network traffic. Performance is better than compute only or compute accel situations. Adding acceleration on the storage-side further reduces latency. SSD scalability is improved by storage offload or acceleration.

With only minimal software change needed to deliver these benefits with storage offloading, system architects should design hardware to support (acceleratable, reduce data, no software change) functions. If possible, this entire group should be supported in future storage-side accelerators. Application designers should also consider offloading these applicable functions.

(acceleratable, data reduction, software change) - Parse. Figure 16 present runtime, network traffic, and SSD scalability for the parsing the required columns from the text version of TPC-H ‘lineitem’ table. In the base case, parsing performance is limited by the network transfer. Storage offload garners network data reduction benefits, addressing the network bottleneck, leaving the general-purpose storage CPU as the performance limit. Acceleration is provided by the UDP (see Section 2.2), that is fast enough to eliminate the storage-side computing bottleneck. The SSD scalability metric shows a similar story, with storage acceleration, both network and computing bottlenecks are addressed, delivering large increase in system capacity and scalability.

(acceleratable, data reduction, software change) - Neural Network. We consider the cases where MobileNetV2 is used to filter relevant images from an image collection as in [41]. Storage acceleration performance is modeled on the edge TPU [7]. Figure 17 presents the runtime elements, network traffic and SSD scalability for the inference process over 1,000 images totaling 10GB.

For offload to the storage CPU, the inference cost dominates, so little benefit is achieved. The TPU acceleration improves inference performance, allowing the intelligent filtering benefit of data reduction to benefit system scalability.

Thus, the functions of (acceleratable, data reduction, software change) should be the second candidates for hardware architect to consider for the hardware support. Significant performance benefits are achievable if software challenges can be solved.

7.3 Can ‘Data Reduction’ justify Storage Offload

We evaluate serialize/archive and select which are (not acceleratable, data reduction, *) to understand whether data reduction alone is sufficient to justify storage offloading.

(not acceleratable, data reduction, no software change) - Select. In Figure 18, we present the runtime elements, network traffic and SSD scalability for the column select on TPCH Q1 ‘lineitem’ table of 10 GB. Because select does little computation, its performance is limited by network transfer. Storage offload provides data reduction benefits, addressing the network bottleneck and both performance and scalability improves.

(not acceleratable, data reduction, software change) - Serialize & Archive. We consider shifting serialize & archive entirely to the storage nodes, confining the network traffic inside the storage cluster. Figure 19 presents runtime elements, network traffic, and
Chen Zou and Andrew A. Chien

Figure 19: Serialize & Archive - not acceleratable, data reduction, software change

Figure 20: Compress - acceleratable, no data reduction, no software change

SSD scalability for archiving a directory with 1024 files totalling 10GB.

Because the storage CPU (offload engine) is not as powerful as the compute node and the runtime latency for the offloaded work increases. However, overall system latency decreases as network transfer time drops sharply, due to data reduction. The SSD scalability metric suggests the same: alleviating the network bottleneck by confining the traffic to the storage cluster increases system scalability.

This exemplar shows that functions that are not acceleratable but deliver data reduction are good candidates for storage offloading to general purpose units. Often they are not acceleratable because of their software complexity or lack of compute-intensivity. For these offloading to a weak general purpose engine can still be a net benefit. Thus, application designers should consider these functions for storage offload, provided the accompanying software change can be achieved.

7.4 Can ‘Acceleratable’ justify Storage Offload

We evaluate the compress and cryptography which are (acceleratable, no data reduction, *) in this section to see whether acceleration alone justifies storage offloading.

(acceleratable, no data reduction, no software change) - Compress. Figure 20 presents runtime elements, network traffic as well as SSD scalability for compressing 10GB data with snappy compression using 64KB blocks. The acceleration is delivered by the Xilinx FPGA U200 with the Vitis compression library [17].

Without storage offloading, system performance is limited compute node speed. Offloading compress increases network traffic (as in Section 6.2) since the uncompressed data is transferred to storage for compression. The resulting network bottleneck limits performance compared to the non-offloading CPU-only situation.

(acceleratable, no data reduction, software change) - Cryptography. In Figure 21, we present runtime elements, network traffic as well as SSD scalability for encrypt 10GB data with AES encryption.

Although acceleration delivers slight performance increase, storage-offloading cannot provide network data reduction, so network data transfer remains a bottleneck. So, even with storage acceleration, there is neither a significant reduction of runtime nor increase in the SSD scalability.

Thus, the performance data affirms the recommendations by the classification framework that acceleration alone can not make the case for storage-offloading (acceleratable, no data reduction, *) functions.

7.5 Summary

We summarize the system scalability and performance improvement with storage acceleration or offload (if the function is not acceleratable) comparing to the baseline of ‘compute only’. In Figure 22, scalability is shown at the left, and speedup on the right. The summary shows the power of our classification framework. In both graphs, the functions that are both acceleratable and provide data reduction enjoy the greatest benefits. In contrast, the functions that
Empowering Architects and Designers: A Classification of What Functions to Accelerate in Storage

...do not reduce data achieve barely any scalability or performance improvements. Thus, we confirm the recommendations from the classification framework with the performance data:

- (acceleratable, data reduction, *) applications are good for storage-offload. Data reduction benefits address network bandwidth limits and acceleration lifts processing performance. System architects and application designer should support functions that need minimal software changes, and see the ones that need significant as second candidates
- (not acceleratable, data reduction, *) functions are secondary candidates for storage-offload. The data reduction benefits alone can justify storage-offloading, and weaker general-purpose storage offloading will not degrade the overall performance as these functions are usually of low computation requirements.
- Acceleratable alone cannot justify storage offloading. (acceleratable, no data reduction, *) are not good candidates, because storage offloading may not be the most efficient offloading, and it may even hurt the performance.

In summary, our classification framework establishes a clear roadmap and order from the chaotic research space of storage offloading. This roadmap organizes the diverse function space, providing clear priorities and guidance for the hardware architect and applications to make offloading decisions.

Further, a roadmap could be derived from the framework to attack the general storage offloading problem. Storage system architects should first focus on the applications that are both acceleratable and produce data reductions, as accelerators targeting functions offloaded to storage could be designed to improve both the functionality and performance of the storage systems. For application and software architects, it’s also clear that there are issues including resource management, service-level agreement as well as addressability because of storage resource sharing, variable storage offload performance, and disaggregate execution environments respectively that need to be solved before widespread adoption is feasible.

8 DISCUSSION AND RELATED WORK

Pioneering work includes “Active Storage” in the 1990s that focused on rotating hard disks in a single system, not the modern context of cloud shared storage services with high-performance flash SSDs [18, 42]. More recent efforts study acceleration systems and assess their performance benefit [28, 31, 32, 34, 39, 40, 43, 45, 46, 49]. Such studies showcase the benefit of a specific acceleration implementation for a set of applications, but give little insight as what makes computation suitable for acceleration, and the benefits likely to accrue - the focus of our classification.

Storage-side Offloading for Specific Workloads Many studies focus on storage-side acceleration for specific workloads. These include exploiting early filtering in OLAP databases [23, 34, 39, 49], offloading compression or deduplication for file/storage systems [19, 20, 32, 38, 48] and offloading new applications such as graph analysis [36] and deep neural networks [41]. Specific benefits include reduced CPU and network loads. Our case study reaffirms these results for OLAP databases, but our classification goes further, assessing a much broader range of computations for offload.

Storage-side Offloading Frameworks Several have proposed frameworks for storage-side offload [28, 35, 43, 44], supporting programmability to enable the development of new offload function implementations. If such use emerges, it could guide hardware customization. This promising work is orthogonal to our work. As short of implementation, it provides no input on how to systematically decide what functions are worthwhile to offload.

Higher-level Storage Interfaces A number of research and commercial efforts have constructed higher-level storage interfaces. Notable amongst these is the idea of a native key-value, storage-side accelerated system. Key-value covers an important subset of storage use [22, 32, 33, 37]. Significant benefits can accrue, particularly when synergies with media management (e.g. flash translation layer) can be exploited [37], and by implementing key-value functions in specialized hardware [32]. Our work addresses a broader scope – the classification can be applied to functions that exist above or below a key-value interface, and to completely different storage stacks.

Cloud Application Refactoring Cloud applications and services such as S3 Select, Glacier Select, and Amazon Aurora [12, 47] refactor the data stack, shifting function to the storage layer. Aurora pushes reliability and synchronization work for high-availability OLTP databases into the storage layer, performing log replay on storage nodes. These refactorings deliver significant performance improvements, showing the promise of storage acceleration. However, software and hardware design of these systems are not available so a direct comparison is not possible.

9 SUMMARY AND FUTURE WORK

We performed a storage acceleration case study using a complex raw data analytics system, showing that large performance and scalability benefits accrue. The greatest benefits come from offloading regex and filtering operations that reduce the quantity of data – both for network transfer and for subsequent query logic processing by the CPU.

Building on these learnings, we studied a broad range of research proposals for function for storage acceleration, producing a simple classification that enables rapid, easy assessment of a computation function. This classification organizes the function space and provide offloading recommendations to storage architects and application designers. It also derives a roadmap to attack the general storage offloading problem for large adoptions.

Interesting extensions include case studies of existing application systems, using the classification system to analyze their performance – and find new opportunities for benefit. Another would be to define general requirements for hardware offload engines based on the classification with the goal of achieving the generality of FPGA’s without the associated complexity and implementation inefficiency.

10 ACKNOWLEDGEMENTS

This work is supported by National Science Foundation Grants CNS-1405959, CMMI-1832230, and CNS-1901466. We gratefully acknowledge support from Intel, Google, and Samsung.