Improving Edge Elasticity via Decode Offload

Ziqiang Feng, Shilpa George, Haithem Turki, Roger Iyengar,
Padmanabhan Pillai[†], Jan Harkes, Mahadev Satyanarayanan

[†]Intel Labs

September 2021
CMU-CS-21-139

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Visual analytics on recently-captured data from video cameras has emerged as an important class of workloads in edge computing. These workloads make intense processing demands on cloudlets, whose elasticity is limited by their smaller physical and electrical footprint relative to exascale cloud data centers. In this paper, we show how cloudlet elasticity can be improved by offloading visual data decoding. We define a new data access API that embodies decode offload, thereby avoiding application-level decoding of visual data. Using thermal, power density and data copying considerations, we identify cloudlet storage as the optimal location for placement of the decode function. Using a proof-of-concept implementation, we show that this approach can lower cloudlet CPU utilization by up to 50–80%, and deliver up to 3.5x improvement in the elapsed time of a typical visual analytics pipeline.

This research was sponsored by National Science Foundation award number: CNS1518865; by United States Defense Advanced Research Projects Agency award number: HR001117C0051; by Intel Corporation award number: A018540296213611; and by a Croucher Foundation Scholarship for Doctoral Study. Additional support was provided by Vodafone, Deutsche Telekom, Crown Castle, InterDigital, Seagate, Microsoft, VMware, and the Conklin Kistler family fund. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: edge computing, video analytics, intelligent storage, active disk

1 Introduction

Edge computing has limited *elasticity,* which is the ability to dynamically increase the supply of computing resources to meet demand. Unlike an exascale cloud data center with an inexhaustible supply of server hardware to activate as load rises, a cloudlet is designed for a modest-sized physical space and electric load. A flash crowd can easily overwhelm cloudlet resources. Elasticity is vital to scalability, because it ensures acceptable queueing delays for cloud-native applications under a wide range of operating conditions.

In this paper, we show how edge elasticity can be improved for visual data analytics, which is one of the "killer apps" of edge computing [1, 2]. A typical setting involves a 24x7 flood of incoming high-resolution data from video cameras, limited bandwidth between cloudlet and cloud, and substantial cloudlet storage that allows data retention on the order of days or weeks [3, 4, 5]. In this setting, the ability to pose *ad hoc* visual queries is valuable. Handling these queries involves re-processing of stored visual data on cloudlets.

We show that the very first step of such re-processing, namely *decoding visual data,* is a surprisingly large burden on edge elasticity. As a solution, we propose the well-known mobile computing technique of offloading computation [6, 7, 8], but apply it to a local rather than remote accelerator. Based on thermal, energy density and data copying considerations, we identify cloudlet storage as the optimal location for placement of the accelerator. Synergistically, we also incorporate batch operations and scheduling into the storage system to reduce stalls and disk seek overheads. We show that this approach can lower CPU utilization on a cloudlet by up to 50–80%, and deliver up to 3.5X improvement in the elapsed time of a typical visual analytics pipeline.

This paper makes the following contributions:

- It highlights the heavy burden of decoding visual data in a new class of visual data analytics applications that are enabled by edge computing and deep learning.
- It defines an API for this new application class that avoids application-level decoding of visual data.
- It shows how this new API can be mapped to an NVMe-attached storage system that combines object-store disks with ASIC-accelerated streaming decode of objects.
- It describes a timing-accurate emulator-based proof-of-concept implementation that we have open-sourced.
- It experimentally validates the performance benefits of this approach on both micro-benchmarks and application-level benchmarks.

2 Background and Related Work

2.1 Attributes of Visual Data

The quantity of visual data captured each year is staggering. In 2017 alone, over one trillion (10^{12}) photographs were captured on smartphones [9]. In 2019, an estimated 300 GB of new data were uploaded to YouTube every minute [10]. Even more visual data is generated by dense IoT camera deployments. Visual data is large but highly compressible, as shown in Figures 1 and 2 for the widely-used YFCC100M [11] and VIRAT [12] datasets. Encoded object sizes

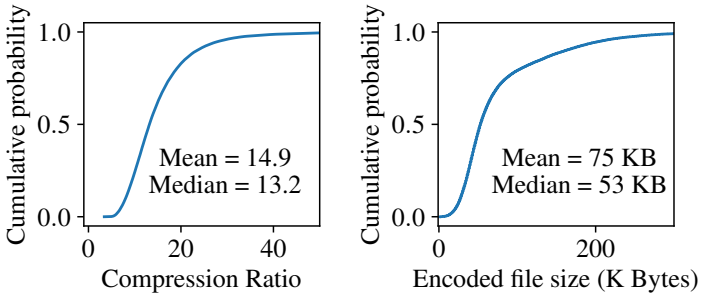


Figure 1: Storage Efficiency of Encoding: YFCC100M

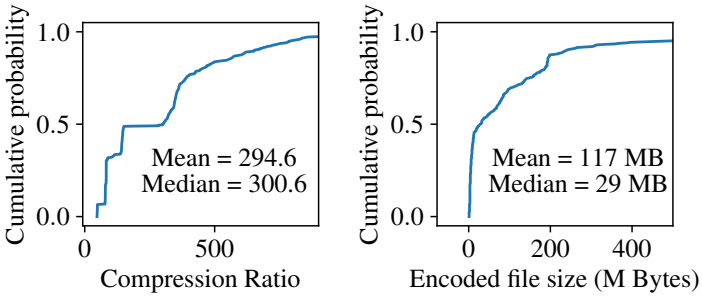


Figure 2: Storage Efficiency of Encoding: VIRAT

will increase over time due to improving camera resolution. As data volumes are high even with compression, the higher capacity and lower cost per bit of spinning disks relative to SSDs make compressed data on disks the only cost-effective storage strategy. Thus, we expect cloudlets to be multi-processor, GPU-enabled machines with multiple direct-attached disks to provide storage and compute capacity for visual data processing.

2.2 Cheap-to-Expensive Forensic Pipelines

Deep neural networks (DNNs) have dominated computer vision since 2012 [13]. Of particular interest to our use case is image forensics, where the user aims to detect interesting events in a large amount of archival visual data. The search criteria are often ad-hoc and context-sensitive, such as "red tour buses from company X" or "a man in blue shirt running." For such unique queries, pre-indexing is unlikely to be useful, and the system has to execute analytics directly on the raw data. Since DNNs are computationally expensive, running them on every single image or video frame can be very slow.

To avoid the high cost of running DNNs on each visual data item, prior work such as NoScope [14], BlazeIt [15], Focus [16], and FilterForward [17] use *early discard filters* that embody a computationally-cheap subset of the full search predicate. Careful choice of this subset can eliminate the vast majority of data items, leaving only a few for expensive DNN processing. In the above examples, "red" and "blue" can be tested with cheap color filters, while "bus" and "man" require DNNs. A pipeline in which the color filter precedes the DNN can have high throughput if most data items are dropped by the color filter. This "cheap-to-expensive" method was originally described in 2004 by Huston et al [18], and is used today by virtually all visual data analytics systems.

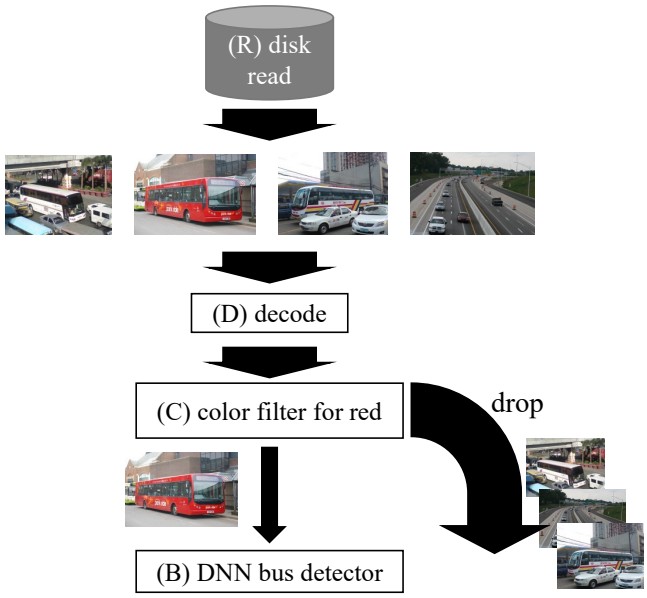


Figure 3: "Search for Red Buses" from BlazeIt [15]

Figure 3 illustrates an example pipeline of "searching for red buses" from the BlazeIt system [15]. Compressed images are read from the disk, decoded, and then examined by a "redness" color filter, which discards images with a below-threshold number of red pixels. The color filter runs several orders of magnitude faster than the bus detection DNN, but is able to drop a large percentage of images that do not match the query. Thus, the pipeline preserves the accuracy of the DNN while delivering high throughput. Figure 4 summarizes the early-discard filters used in this paper. These have been used in the published work mentioned above, and have been proven to be highly effective.

3 Problem: High Cost of Decoding

The analytics pipeline shown in Figure 3 consists of four main operations: (R) reading encoded images from disk, (D) decoding into pixel arrays, (C) execution of color-based filtering, and (B) execution of the bus detection DNN. Figure 5 shows the average per-image CPU cost of these four steps in processing 50,000 images from the YFCC100M dataset. The experiments were run on a cloudlet (4-core slice of a server with two Intel® Xeon® E5-2699v3 processors at 2.30 GHz and an NVIDIA GTX 1080 Ti GPU) with a Seagate 4TB hard disk drive (7200 RPM, SATAv3). This configuration reflects the typical per-drive resources of a 2-socket server with 8–12 direct-attached disks.

In Figure 5, the four steps are added incrementally from left to right. Initially (label "R"), the process is I/O-bound as the read data is discarded immediately. As soon as decoding of visual data is added (label "R+D"), CPU time jumps dramatically. In fact, image decoding (Step D alone) consumes 70% of the CPU cost of the full pipeline. The third step (label "R+D+C") shows that applying a color detection filter on all data items only increases total CPU usage modestly, relative to the cost of decoding. What this implies is that the color filter (Step C) can operate

Frame sampling. Useful when event of interest takes some time [14], e.g., pedestrian crossing street.

Color filter. Counts pixels in given RGB range; can cheaply detect sky (blue), vegetation (green), etc.

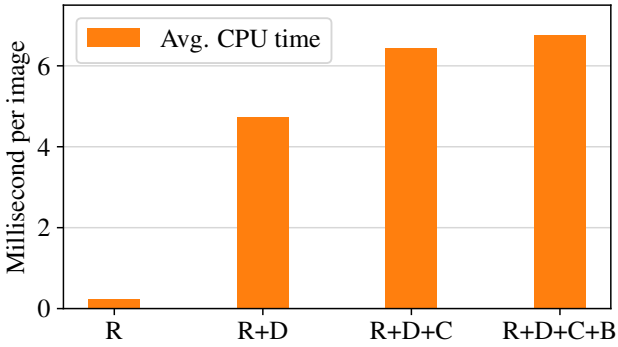
Face detection. Finds faces in images. Computationally expensive, but is an effective early-discard filter when finding human activities or recognizing individuals.

Image difference. Computes mean square error (MSE) between current and prior image; if MSE is small, can assume identical results of later processing stages. Useful in video processing.

Perceptual hashing. Like image difference, but more robust to pixel noise, minor lighting differences, etc.

Tiny DNNs. Much smaller, faster, but less accurate versions of standard DNNs [14, 15]. Useful as early discard filters prior to running an expensive DNN.

Figure 4: Early Discard Filters Used in This Paper



R: Read files from disk
D: Decode JPEG to RGB
C: Color filter for redness
B: Bus detection with DNN

Figure 5: High Scalability Cost of Image Decode

at much higher throughput than JPEG decoding (Step D). The small difference between the bars labeled "R+D+C" and "R+D+C+B" shows the benefit of early discard. The expensive DNN for bus detection (Step B) only has to be applied to about 3% of the images that pass the color filter. We seek a way to reduce cost (D) that is simple, effective and future-proof. Figure 6, which previews our experimental results from Section 7, confirms the effectiveness of the solution described in the rest of this paper. Comparing Figure 6 to Figure 5, we see that the average per-image CPU cost of decoding is reduced to a half (2.3 ms vs 4.7 ms), indicating potential improvement of scalability.

4 Solution: Decode-Enabled Storage

Our solution is developed from an application viewpoint, rather than a systems viewpoint. All that a typical visual analytics application desires is to obtain RGB arrays of the visual data in

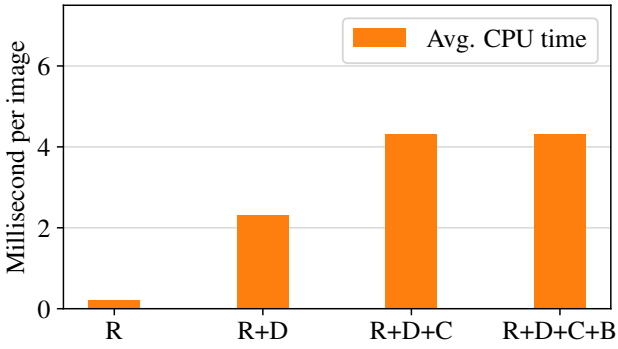


Figure 6: Impact of Our Solution on Figure 5's Workload

its virtual memory. The application does not care exactly how these arrays materialize. This leads to a simple question: *"Why doesn't the storage subsystem return the decoded data when it is read?"* In Sections 4.1 and 4.2 below, we propose a *decode-enabled storage API* that embodies this abstraction. The API simultaneously simplifies application development and allows for placement of functionality close to their optimal position (discussed in Section 5). Our APIs let an application obtain the decoded version of a visual data object stored on the disk. Beyond that, it supports a high-value subset of image processing functions and multi-object read optimization. In Section 5, we consider alternative approaches to implementing the abstraction of decode-enabled storage.

4.1 Extending the Object Store API

Decode-enabled storage extends the well-known object store concept [19] that allows an application to create, read, write, and delete logical objects. Our extensions let the calling application specify operations and transformations to perform on an object as it is read. This is embodied in the `FetchAndDecodeObject` call as follows:

```
FetchAndDecodeObject(
    int64 object_id,
    int32 opcode,
    void* params,
    iovec* where_to_put_decoded_object,
    iovec* where_to_put_original_object)
```

Each object is addressed by an integer Object ID. The `opcode` field indicates whether to fetch the original compressed object, the decoded version, or both. Fetching both is useful in a server context, where images may need to be transmitted across a network after local filtering: decoded objects improve analytics performance and reduce CPU load, while the original versions can be sent without re-encoding. The final two vectors indicate memory regions into which the fetched data will be placed using a scatter-gather approach.

Other operations may also be requested using `opcode`. In this paper, we focus on image decoding and (content-based) cropping (e.g., face detection), which can be accelerated with ASICs

and will remain useful long into the future. Parameters for operations can be provided through the `params` pointer.

Partial read or write of an object is not supported. This enables additional disk optimizations, reduces internal fragmentation, and maximizes sequential reads. These semantics are a natural fit for compressed image data, but work well for video, too. For example, B-frames in H.264 are encoded using information from both past and future frames. Thus, to successfully decode a single frame, the decoder needs to retrieve large amounts of surrounding data, possibly the whole object. We suggest keeping the size of individual objects moderate, on the order of several MBs. Very large videos (GBs to TBs) can be stored as a sequence of objects, for which the mapping can be stored in a separate object.

4.2 Multi-Object Batch Iteration

The above single-object API can be seen as offloading computation (e.g., decode) at the granularity of an image. This granularity can be increased to reap additional benefits. Visual data analytics is typically a form of batch processing, applied on thousands to millions of images, with no requirements on order. Performance can be improved by re-ordering the objects, which, for example, reduces disk seeks or exploits parallel read heads. Although it is well known that access pattern can have a great impact on I/O performance, the most efficient read order varies from device to device. On traditional disks, it primarily depends on physical block location and disk geometry, but this can be obscured by logical block addressing and remapping. On object store disks such as Seagate's Kinetic HDD [20], it is further complicated by the firmware's approach to storing, fetching, and caching metadata. Access to this information is obscure in the application, but straightforward inside the disk. Moreover, a system may use a heterogeneous set of disks that further complicates application-level optimization.

Therefore, it is more intuitive to optimize this within the storage subsystem, behind a unified batch-oriented API call. Note that this is different from request scheduling on current disks, which only re-order requests on short work queues. Our decode-enabled storage API provides an iterator-style batch operation as follows:

```
IterateCollection(
    int64 collection_id,
    int32 opcode,
    void* params,
    int64 logical_index,
    int64* flags,
    int64* returned_object_id,
    iovec* where_to_put_decoded_object,
    iovec* where_to_put_original_object)
```

The list of Object IDs to fetch are created *a priori* in an object of some custom format, referenced by `collection_id`. The application iteratively calls this API to retrieve another object. The `returned_object_id` tells which object was fetched. The API guarantees exactly-once semantics of objects — exhaustive and non-repeating, but makes no promise of the order. The `logical_index` tracks the iterator cursor. A flag `COLLECTION_LAST` is set in `flags`

Architecture	Efficiency (GFLOP / J)
CPU (Core i7)	1.14
FPGA (Xilinx LX760)	3.62
GPU (NVIDIA GTX285)	6.78
GPU (AMD R5870)	9.87
ASIC	50.73

Source: Table 4 in Chung et al [21]

Figure 7: Energy Efficiency of Hardware Accelerators

to terminate iteration when the last object is returned. The other fields are the same as in the single-object API.

5 Implementation

The API described in Section 4 discloses application intent to the system. Using a hardware accelerator to implement the most compute-intensive parts of this API is clearly the way to reduce CPU demand, and thereby improve scalability. On cloudlets without the accelerator, a pure software implementation of the API can provide compatibility. Applications can be written to this API today, and will continue to work unmodified over the long period of time that it typically takes for hardware optimizations to gain market share. This is the same strategy that has been used for GPU acceleration in popular open-source libraries such as OpenGL and DirectDraw.

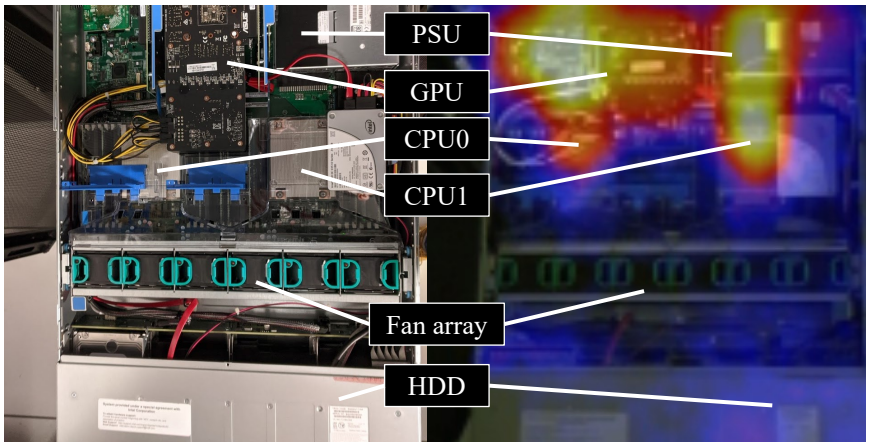
Two questions follow from this decision. First, what type of accelerator should we use? Second, where should it be placed? We discuss below the design rationale that leads to our recommended solution of ASICs within storage devices.

5.1 Energy and Thermal Considerations

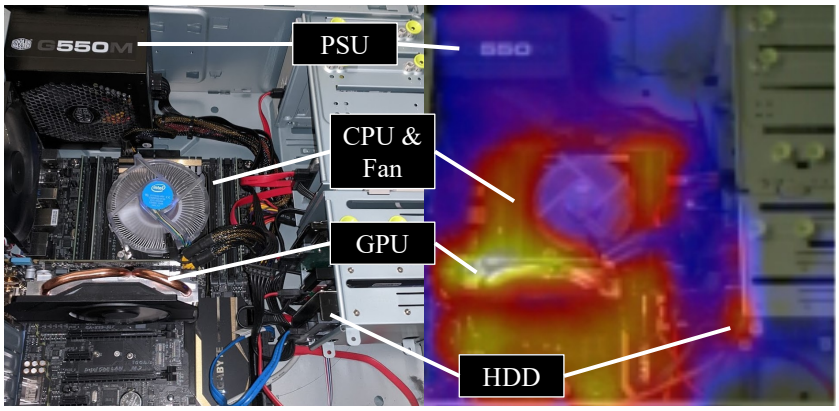
As mentioned in Section 1, energy efficiency is a key consideration for a cloudlet. This is especially true for a hyperconverged multi-tenant cloudlet that must fit into very limited space, yet support many CPU cores and at least one high-end GPU for visual data analytics. The tight thermal envelope and limited electrical power budget of such a cloudlet requires very careful attention to energy and cooling efficiency.

Figure 7 from Chung et al [21] shows the relative energy efficiency (expressed in GigaFLOPs per Joule) of different hardware accelerators. These specific measurements are for matrix multiplication, but the trend holds across applications [22]. Energy efficiency improves as one moves down the rows of Figure 7, but comes at the cost of flexibility. When flexibility is paramount, CPUs are optimal; when it is not important, ASICs are optimal. Intermediate points in this spectrum correspond to programmable accelerators such as GPUs.

In our setting, flexibility is not important. The formats used by visual data such as JPEG, PNG, JPEG2000, and MP4 are well standardized today. These will remain unchanged forever, because of vast archives of precious data stored worldwide in these formats. If new formats arise in the future, support for them can be software-only on legacy cloudlets. New accelerators can support the new formats, in addition to all the old formats. Use of the APIs described in Section 4 insulates legacy



(a) Rackmount Cloudlet



(b) Standalone Cloudlet

Figure 8: Thermal Heatmap of Typical Cloudlets

applications from this discontinuity in hardware implementations. They only deal with decoded data, and are thus impervious to lower-level changes. What is unlikely to ever change is the need for decoding data as the first step in visual data processing pipelines.

ASICs, which are fixed-function (i.e., non-programmable), thus emerge as the best type of hardware accelerator to use. Their lack of versatility is not a handicap for our use case, and their superior energy efficiency is a major advantage.

The placement of the ASIC is also guided by energy and thermal considerations. Figure 8 shows the thermal heat maps of a typical rackmount cloudlet and a typical standalone cloudlet while they are processing a visual data pipeline. The brightest areas (in white, yellow and orange) represent the current "thermal bottleneck" of the system. Adding new hardware to any of these areas will only worsen this bottleneck, and make cooling the system more difficult. The power density in these areas is already high, and delivering more power there will also be difficult. As long as performance is not compromised, adding new hardware to the coolest parts of this heat map (in blue or black) that are furthest from the current thermal bottleneck is the wise path to follow. Storage devices are among the coolest parts of Figure 8, and located furthest from the thermal bottleneck. They also represent the starting points of all visual data pipelines. They are thus the logical choice for placement of the ASIC.

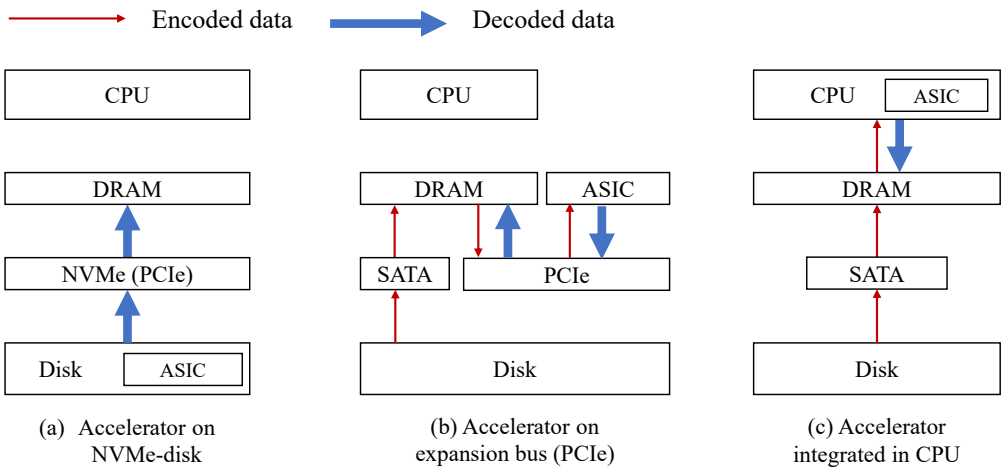


Figure 9: Alternative Placement of Decode Accelerator (ASIC)

5.2 Minimizing Data Copying

A well-known design principle for scalability from "big data" systems and database systems is to minimize data copying [23, 24, 25]. Locating the decode-acceleration ASIC in a storage device aligns well with this principle. No new data copies are needed; rather, data is decoded in a streaming operation as it is read off the disk surface. This may seem to be a counterintuitive optimization at first glance, because early decoding greatly increases the bandwidth demand on the SATA interconnect from disk. However, this can be overcome by replacing SATA by the modern NVMe host-storage interconnect [26]. NVMe was originally created to support the much higher bandwidth demand of SSDs. There is industry speculatation that NVMe will become the unified interface for all storage types in the near future, including SSD and disks [27]. By fortunate coincidence, this trend aligns well with our proposal to place decoding functionality on disks.

Figure 9(a) illustrates the placement of a decode ASIC directly on disk. A cloudlet can be attached to multiple such disks to exploit parallel storage bandwidth and decode throughput. In other words, compute and storage scale together when one adds more disks to a system with this configuration.

An alternative approach, shown in Figure 9(b), is to separate the ASIC from storage and place it on the I/O bus (typically PCIe). Similar to (a), this strategy makes it easy to add more accelerators to the system. However, it has at least two deficiencies relative to Figure 9(a). First, it requires some host mediation of the decoding process, incurring context-switching overheads. Specifically, the host must initiate reads of encoded objects from disk into DRAM; once complete, it triggers the decoding of these objects and sends them to the accelerator. While the overhead of this mediation can be reduced by batching over many objects, it can never be reduced to zero. Second, it incurs two more round trips of the encoded data over the system bus. Assuming a compression ratio of 15, this amounts to $2/15 = 13\%$ additional cost in terms of bus data transfer and energy. Decode in GPU (e.g., NVIDIA's NVDEC) is an instantiation of this strategy, but GPUs are more expensive and less energy-efficient than an ASIC, and are already a thermal bottleneck (Figure 8).

A third alternative, shown in Figure 9(c), avoids data copying by integrating the ASIC directly on a CPU die. An example of this approach is Intel's Quick Sync Video (QSV) feature. With this approach, decoded data completely bypasses the system bus and possibly even DRAM (if it can

SATA	500 – 700 MByte/s
NVMe	1,000 – 6,000 MByte/s
HDD "internal"	100 – 300 MByte/s
SSD "internal"	$>$ 500 MByte/s

Table 1: Host-Device Bus Technologies and Storage Devices Internal Read Throughput

be written directly to CPU cache). However, as discussed in Section 5.1, there are strong thermal considerations that argue against adding an accelerator to the already-hot CPU die if an alternative placement can perform just as well. In addition, it is difficult to scale the CPU-integrated decoder. Most processors have many CPU cores, but just one (if any) hardware decoder block. Such a decoder typically can handle hundreds of frames or images per second, more than sufficient for the vast majority of real-world use cases. Thus, there is little incentive to add more than one such decoder to a general-purpose processor. However, as our experiments in Section 7 show, we may need many times the decode capability to fully utilize all of the compute cores and disk bandwidth in a reasonably-sized cloudlet. Finally, a CPU-integrated solution provides a fixed decode capability that cannot be scaled up easily (adding processors or additional servers is not feasible in a cloudlet context); in contrast, a solution with one appropriately-sized ASIC decoder per disk will naturally scale up decode capability as more storage is added to a system.

5.3 Technical Feasibility

Decoding on disk critically depends on the use of NVMe host-storage interconnects. Table 1 lists reference speeds of SATA and NVMe, as well as the internal transfer speeds of disks and SSDs. With an average compression ratio of 15x for JPEG, a disk that delivers 200 MB/s from its platters will produce 3000 MB/s of decoded data. This is well above what SATA can sustain, but well within the bandwidth supported by current NVMe products [28].

Fixed-function hardware and FPGAs have already been adopted on hard disks for other functions [29, 30, 31, 32]. Low-power, low-cost hardware accelerator or FPGA for image decoding has been studied, prototyped, and validated in both academia and industry (e.g., [33, 34, 35, 36]). Table 2 compares the published performance of two FPGA-based JPEG decoders with experimental measurements of software decode on a single host CPU core and an embedded CPU core of an active disk (ARM Cortex A53 on Seagate Kinetic HDD). For reference, we also benchmarked with Intel Quick Sync Video (QSV), a hardware-accelerated decoder integrated on certain Intel processors. We see large performance gains with accelerators compared to software decode on CPUs.

Unlike many previous designs of *intelligent storage* or *active disk* [37, 38, 39, 40, 41, 42, 43], our design precludes the execution of arbitrary code within a disk. This greatly reduces the disk's cost, complexity, and security risk. In addition, it avoids putting the cost-, power-, and memory-limited on-disk computational capabilities in competition against well-provisioned host processors, which are designed for computational throughput. Any advantage of on-disk general computation is often rendered obsolete by rapid improvements of the host system driven by Moore's Law. Decoding, in contrast, represents a very constrained form of application-level logic that is well standardized, and is unlikely to become obsolete even if the workload evolves in the long term.

Device	JPEG Decode Speed
Disk CPU (ARM-based, 1.0 GHz)	15 MPixel/s
Host CPU (Intel-based, 2.3 GHz)	60 MPixel/s
FPGA 1 [36]	73 MPixel/s
FPGA 2 [35]	140 MPixel/s
Intel Quick Sync Video	600 – 1,000 MPixel/s

Table 2: JPEG Decode: Software vs HW Acceleration

Another advantage of on-drive decoders is that they can be co-designed with the disk's specs in order to fully utilize its internal throughput and outbound bandwidth. A decode-enabled disk can be equipped with multiple hardware decoders. Under a simple system model, we can bound the number of decoders by the minimum between its internal (encoded) object throughput and outbound (decoded) object throughput:

$$\frac{\min\left(\frac{\text{Disk Internal Read Speed}}{\text{Avg Encoded Object Size}}, \frac{\text{NVMe Speed}}{\text{Avg Decoded Object Size}}\right)}{\text{Object Throughput Per Decoder}} \tag{1}$$

The arguments for images apply similarly to video decoding. IP cores exist that can decode 4K resolution H.264 video at 60 – 120 frames per second (FPS) [44, 45]. We will explore how much on-disk decoding capability is needed to deliver performance gains in Section 7.

5.4 Beyond Decoding on Drive

Beyond hardware-assisted decode, it is difficult to justify running computer vision operations on a disk CPU. Computer vision algorithms tend to be both compute- and memory-intensive, and have been changing at a rapid pace. They are more suited to execution on powerful host CPUs or GPUs. General compute capacity in drives is typically modest, and it is usually not worthwhile to implement new hardware for rapidly evolving workloads.

The one exception we consider here is image cropping on disk. Concretely, we consider two types of cropping. In the first type, the application provides a bounding box. Cropping based on pixel coordinates incurs data movement, but only trivial computation, and can be performed efficiently on disk. This is particularly useful for static cameras where the application has *a priori* knowledge about regions of interest, e.g., portion of a traffic camera view covering a crosswalk [17].

The second type uses dynamic coordinates based on image content analysis. While this can be expensive in general, we consider operations that can be accelerated through hardware, for example, face detection. Prior research on face detection accelerators achieved between 30 and 600 FPS [46, 47, 48, 49, 50]. These accelerators output the coordinates of the faces, which the disk processor uses to perform cropping.

Cropping is possible only after decoding, but offers the opportunity to reduce the amount of data transferred on the bus. Only the cropped patches are returned, and the other bytes are discarded. This has the potential to reduce the bandwidth requirement, depending on the crop size or sparsity of faces in the data set.

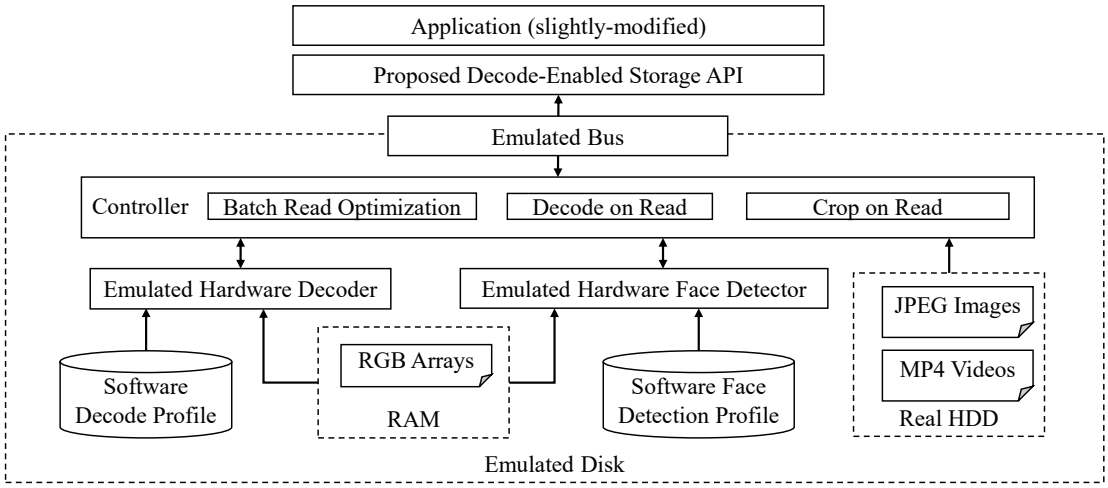


Figure 10: Decode-Enabled Storage Emulator used in Experimental Evaluation

Although we only consider cropping here, as other computer vision algorithms become standardized and implemented in low-cost hardware, they can be added to the list supported by a decode-enabled storage device.

6 Timing-Accurate Emulated Prototype

We implement *timing-accurate emulation* of an NVMe-attached decode-enabled disk that allows execution of real application code and measurement of wall-clock time and real OS-level statistics (e.g., CPU time, bandwidth). The emulation allows us to experiment with a wide range of parameters that are otherwise unavailable in existing hardware products (Section 7). This section describes our discrete event-driven simulation that uses pre-computation and modeling to emulate HW decoders, while disk timing is based on a real HDD. [1]

6.1 HW/SW-based Emulation Framework

Figure 10 depicts the architecture of our emulator, implemented in a similar way to DiskSim [51] and Memulator [52]. Application programs are largely unchanged except for the use of the new APIs to fetch (decoded) data. It includes critical components of a decode-enabled storage device – the host-disk interconnect bus, potentially parallel HW accelerators for decoding, and mechanical disk timings. The controller implements the logic to control data flow and coordinate different components in order to service a request from applications.

For each software-emulated component, we construct (1) a model to calculate the (content-dependent) completion time for operations, and (2) a mechanism to generate the actual results produced (e.g., the actual decoded pixel arrays). The latter mechanism must execute faster than the modeled time; this is achieved by serving pre-computed results from memory. As this generally runs faster than needed, the system inserts high resolution sleep to achieve the modeled latency.

[1] Our emulation and evaluation code is available at `https://github.com/fzqneo/smartssd-image`

Following DiskSim, we use a discrete event-driven approach to model complex interactions between components. For example, requests to the decode ASICs are serialized through a priority queue sorted by the simulation timestamp. Thus, requests are ordered "correctly" even if generated out of order by the concurrently executing components. The simulation clock is continually updated to match the real world time. When a request's computed completion time is reached by the simulation, we send the response back to the application.

6.2 Emulating Specialized Hardware

Prototypes ASIC- and FPGA-based image decoders [33, 34, 35, 36] are typically characterized by a metric specified in MegaPixels per second (MPixel/s). Hence, we parameterize our emulated decoder with a targeted MPixel/s. With RGB format, 1 MPixel/s is equivalent to 3 MByte/s of output. In practice, decoding time may vary based on content and compression level of images. Our emulator accounts for such variability, while maintaining a target speed. We compute a global scaling factor that scales the average software decode time for an entire image dataset to the target rate; we apply this factor to the software decode time of each image to obtian its simulated HW decode time. More concretely, suppose the data set has N images, the software decode time of image i is t_i^{sw}, and its decoded size is r_i MPixel. When simulating an image decoder parameterized at M^{sim} MPixel/s, the simulated decode time of image i is calculated as:

$$t_i^{sim} = \frac{\sum_{k=0}^{N} r_k}{\sum_{k=0}^{N} t_k^{sw}} \cdot \frac{t_i^{sw}}{M^{sim}}$$

To emulate real-time hardware decode, we pre-decode all images and store them in a ram-disk. At run time, the decoded data is rapidly returned to the application.

We emulate video decoding and face detection hardware in a similar fashion. For simplicity, we parameterize them using a target frame per second (FPS), and scale individual elapsed times based on profiled software times.

6.3 Emulating Disk Hardware Timing

We emulate mechanical disk timing factors, such as seeks, platter rotations, and block cache by reading files from a real hard disk. Although this will include overheads of the OS, filesystem, and bus, we find that these are small by performing similar tests on a fast SSD. We were careful to clear the OS page cache before each experiment.

6.4 Emulating the Bus

We parameterize the emulated bus by a maximum bandwidth (MByte/s). Each (decoded) object exclusively occupies the bus during its transmission and other objects must wait for the bus to be relinquished. We assume that the bus operates at the maximum rate when in use, and is idle otherwise. This simplified model is sufficient to estimate transfer rates for different bus technologies.

Host	
CPU (`cgroup`)	4 cores/8 threads, 2.30 GHz
DRAM (`cgroup`)	64 GB
GPU	NVIDIA GTX 1080 Ti
Decode-Enabled Disk (Emulated)	
Host-Disk Bus	2,000 MB/s
HW JPEG Decoder	140 MPixel/s $\times 5$
HW Face Detector	30 FPS $\times 1$
HW Video Decoder	480 FPS @ 720p
Standard SATA Disk (Real, baseline)	
Specs	3.6 TB, 7200 RPM, SATAv3
Throughput	Bulk: 187 MB/s; JPEG: 98 MB/s

Table 3: Default Experiment Setup and Parameters

7 Evaluation

We ran experiments on a workstation with two Intel® Xeon® E5-2699v3 processors (total of 36 cores/ 72 threads @ 2.3 GHz), 128 GB DRAM, and an NVIDIA GTX 1080 Ti. Because our emulator pre-stages decoded data in DRAM to emulate fast HW-accelerated decode, we randomly sampled 50,000 images from the YFCC100M data set [11]. The corresponding decoded data totals 51 GB, fitting comfortably in DRAM. Likewise, we sampled 6 videos from the VIRAT Release 2.0 Ground data set [12], which are encoded in H.264 format at 1080p / 720p @ 30 FPS. We ran real visual analytics application code, adapted from published visual analytics systems [14, 15] and written in OpenCV, TensorFlow, and PyTorch. The emulator was run on a different NUMA node than the application programs to avoid interference.

Table 3 summarizes the setup and default parameters used in experiments. To saturate a 2,000 MB/s bus, we used Formula 1 to calculate that we need about 5 JPEG decoders at 140 MPixel/s each (ref. Table 2). We first report results under these default settings, and then study the effect of varying different host and disk parameters. The preview results presented earlier in Figure 6 hinted at the potential for greatly improved scalability based on CPU utilization, here we delve into a more detailed study based on wall-clock time, disks supported per server, and end-to-end throughput, by addressing the following questions:

- Can decode-enabled storage improve application-level performance metrics?
- Can decode-enabled storage reduce processing load on the cloudlet CPU?
- Is NVMe necessary and sufficient for transmitting decoded data from the disk?
- How much processing is needed on the disk?
- How many disks and accelerators can be connected to a cloudlet before saturating the host system's resource?
- How does decode-enabled storage compare to alternative solutions to reducing decode overhead?

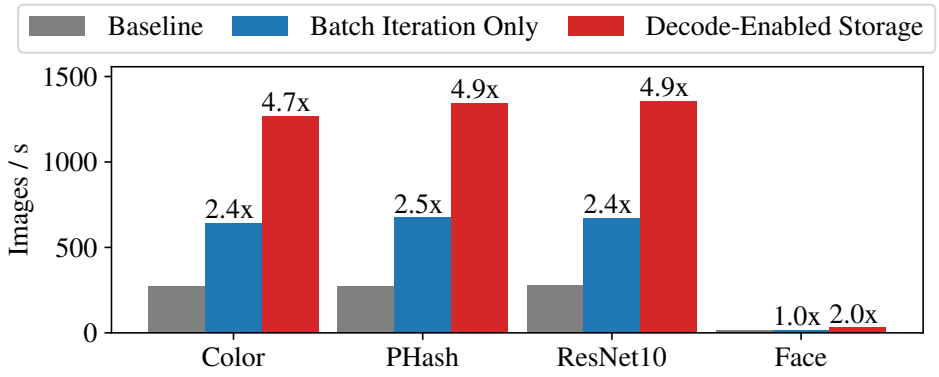


Figure 11: Application Throughput of Micro Benchmarks

7.1 Micro Benchmarks on Images

We first evaluate a series of micro benchmarks of the initial early-discard filters of partial analytics pipelines. As described in Section 2, such filters are effective in quickly dropping irrelevant data and greatly reducing per-image computation cost. We run the following on our dataset of 50K images from YFCC100M:

- `Color` finds images with many red pixels.
- `PHash` calculates an image's perceptual hash value.
- `ResNet10`[15] is a tiny DNN based on ResNet [53], with 65×65 input and 10 layers (reduced from 224×224 input, 50–100 layers).
- `Face` detects faces in an image, annotates the bounding boxes, and drops images with no faces.

`Color` and `PHash` offload image decode operations to the disk. `ResNet10` offloads image decoding to the disk, runs resizing and normalization on the CPU, and runs the neural network on the GPU. `Face` offloads both image decode, face detection, and cropping to the on-disk accelerators. Only the list of cropped patches containing faces, or a null list if there are none, is returned.

Effects on Application Throughput

Figure 11 reports the benchmark throughputs (image/s) for three systems: (a) Baseline: uses standard SATA-connected disk and software decode; (b) Batch Iteration Only: optimizes multi-object batch read order (Section 4.2) on a standard disk; (c) Decode-enabled Storage: combines batch iteration, on-disk decode, and NVMe. The data labels show improvement factors relative to Baseline.

Batch Iteration Only is approximated by accessing files sorted by the starting blocks of the file extents returned by Linux system call `FIEMAP`. This does not fully account for a hardware implementation, but provides a partial estimate of the potential gain. This optimization mainly improves wall-clock time, but slightly improves CPU cost as well.

We see batch iteration alone is effective (up to 2.5x improvement) by improving I/O efficiency, but is not responsible for all performance gains. Adding on-disk decode HW and NVMe achieves

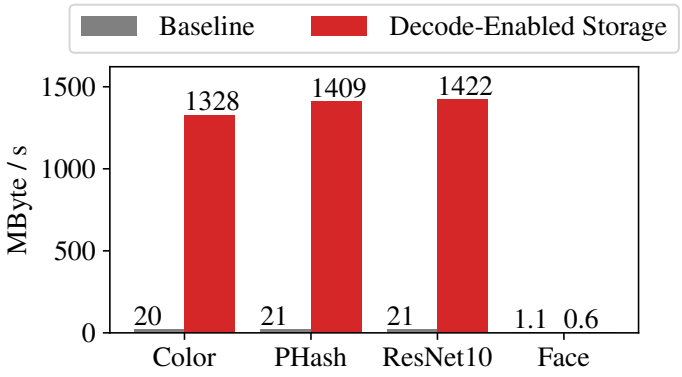


Figure 12: Data Transfer Rate on Disk Bus

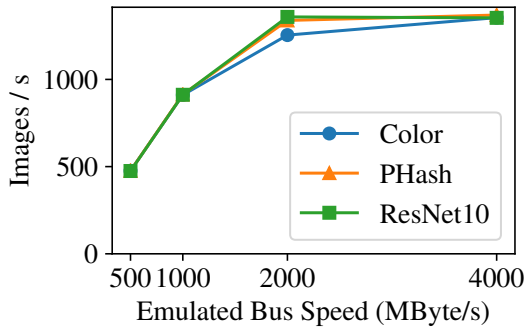


Figure 13: Effect of Bus Speed

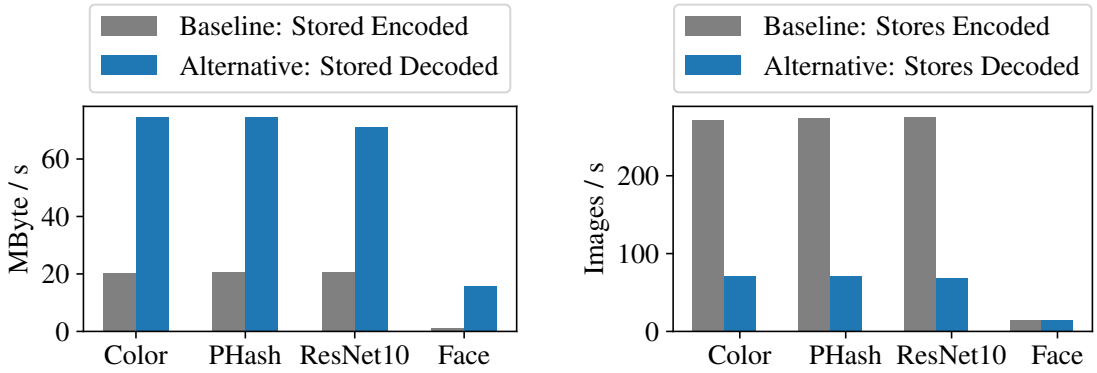
up to 4.9x improvement over Baseline. `Face` is much slower than the others, because face detection is computationally expensive. However, even a single face detection chip at 30 FPS delivers 2x gain over software detection on the 4 CPU cores used in this experiment.

Is NVMe Necessary and Sufficient?

Figure 12 measures the data transfer rate on the bus for Baseline and Decode-enabled Storage. With decode-enabled storage, up to 1,400 MB/s is transferred to the host, clearly exceeding SATA bandwidth. This increase reflects two factors: (1) the transfer of decoded, rather than compressed images; (2) the host CPU, freed from decode tasks, can process images at higher throughput (Figure 11).

With `Face`, decode-enabled storage actually consumes less bandwidth than baseline (0.6 vs. 1.1), despite running 2x faster (recall Figure 11). This is due to cropping-on-disk (Section 5.4) that only sends cropped faces. In YFCC100M, only 23% of images contain human faces, with an average size of 97.5×126.9 pixels. Hence, even though they are sent as uncompressed pixels, the face crops require less bandwidth than whole-image JPEG files for the entire dataset.

To study how the bus speed impacts performance, we throttle the emulated bus's bandwidth between 500 MB/s (SATA speed) and 4,000 MB/s (high-end NVMe speed), and measure throughput for `Color`, `PHash` and `ResNet10` (Figure 13). We see that bus bandwidth has



(a) Transfer Rate on Bus

(b) Application Throughput

Figure 14: Effect of Storing Decoded Images

almost linear impact on application throughput up to 2,000 MB/s, again confirming the importance of NVMe for our design.

Alternative Solution: Storing Decoded Data

An alternative approach to completely eliminating the decode cost is to store the decoded data directly on the disk, at the expense of reading more data from the disk platters. Figure 14 reports the host-disk transfer rate and application throughput of this approach. As expected the data transfer rate increases significantly by up to 3.7x over baseline, limited by the internal read speed of the disk (Table 1). This increase is due to larger sequentially accessed files, resulting in fewer seeks, and the fact that the decode bottleneck is removed from the CPU. Unfortunately, this increase is offset by the 15x inflation in object sizes, resulting in a net decrease in application-level throughput. Furthermore, this in effect reduces the disk's capacity by 15x as well. Overall, storing decoded images on the disk is a losing proposition due to poor performance and poor cost-efficiency.

Scaling on A Large Cloudlet

In a realistic edge deployment, a cloudlet typically has dozens of CPU cores that process data from a dozen disks in parallel. Among others, *general-purpose* compute cycles are a precious resource. Utilizing decode-enabled storage is a more economic way to improve elasticity than adding more CPU hosts. This leads to a question: *"For a given cloudlet, how many disks should be used?"*

We first investigate how many CPU cores the application utilizes when saturating a single disk's throughput. Figure 15 shows the application throughput on a single decode-enabled disk, as we vary the number of physical cores allocated to the processes. The "kinks" of the curves indicate when the CPU cores start being underutilized, as the bottleneck shifts to I/O. For `Color`, `PHash`, and `ResNet10`, the sweet spot appears to be 4 cores / 8 threads. `Face` is not limited by CPU, as the processing running on CPU is negligible.

Based on measured CPU/GPU utilization at that point of saturation, we extrapolate the scaled-out performance as more standard or decode-enabled disks are connected to our 36-core cloudlet. This machine has 136 GB/s DRAM bandwidth, well above the bandwidth demands calculated in this scenario. Besides, we benchmarked the GTX 1080 Ti GPU can run `ResNet10` at

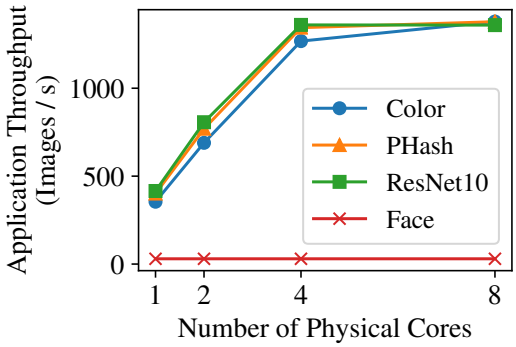


Figure 15: Effect of CPU Cores on Throughput

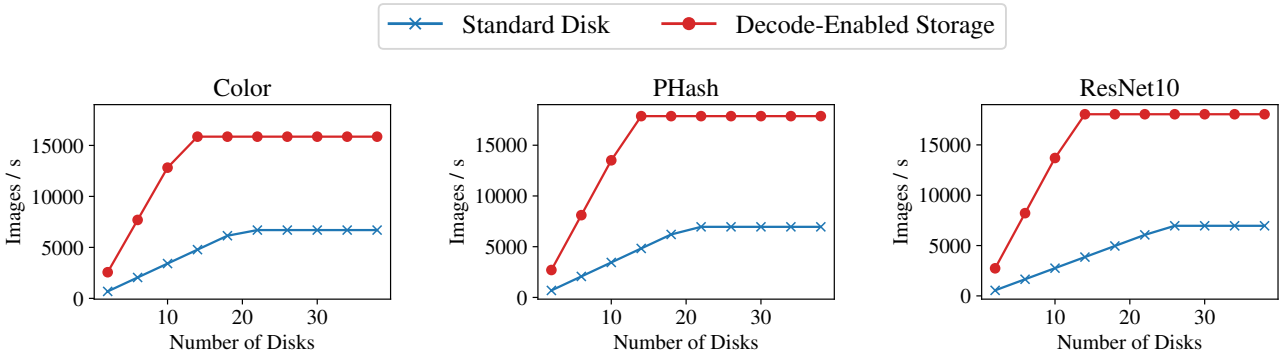


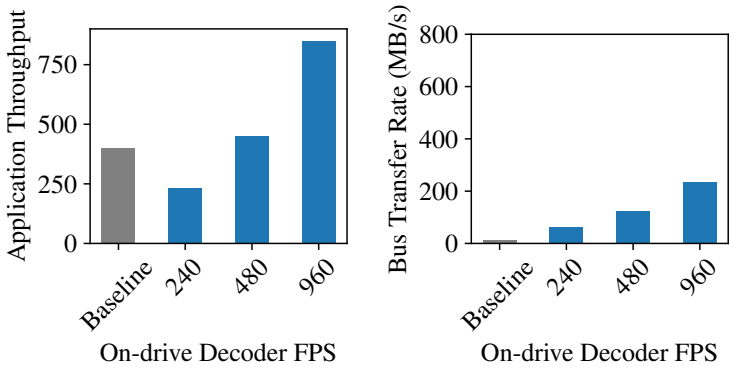
Figure 16: Extrapolated Application Throughput with Varying Number of Disks

40,000 images/sec, and should not be a bottleneck in our scaling range. Figure 16 shows the extrapolated application throughput as the number of disks is increased. For all cases, decode-enabled storage achieves more than 2x higher throughput than standard disks. It also suggests connecting up to 20 decode-enabled disks to a cloudlet, requiring up to $20 \times 1.5 = 30$ GB/s of data transfer rate. Can a modern machine support this required I/O bandwidth? With 40 lanes of PCIe 3.0 per socket, and two sockets, the benchmark machine has a theoretical peak I/O bandwidth of 80 GB/s, so it should be feasible to support more than 20 decode-enabled disks in one system.

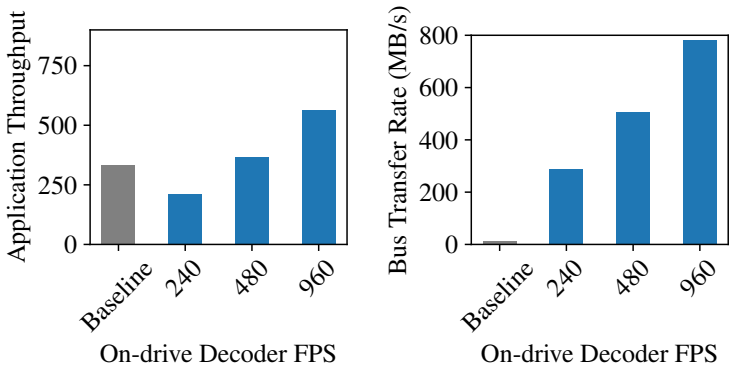
Figure 16 also shows that the general-purpose CPU cores, when freed from the decode task, can execute the early-discard filters at more than 15,000 image/s. This is approximately 15–20x higher than the JPEG decode throughput we can obtain from the Intel Quick Sync Video accelerator (Table 2). In other words, on-die accelerators need to be scaled up by 15–20x to meet the whole CPU's processing speed. The thermal, engineering, and business challenges associated with this were discussed in Section 5.

7.2 Micro Benchmarks on Video

We run experiments on the VIRAT video data set using the pipeline from [14]. Here, we sample video frames at fixed intervals; then compute the mean squared error (MSE) between the sampled frame and the frame one second (30 frames) earlier. If MSE is lower than a threshold, we consider the frame to have the same content as the prior one and suppress further processing. The baseline



(a) Sample 10% of Frames (every 10 frames)



(b) Sample 50% of Frames (every 2 frames)

Figure 17: Video Decode on CPU (Baseline) vs ASIC

solution decodes video in software as fast as it can when running the pipeline. With decode-enabled storage, decoding is performed on the disks and skipped frames are not transmitted. MSE computation always runs on the host.

Processing video shows two key differences from processing image. First, because of the very high compression ratios and large file sizes of video, disk read is not as much of a bottleneck as with images, and the use of batch iteration makes little difference. Second, the frame sampling rate greatly affects the relative cost of decode. When frames are sampled more frequently, more CPU cycles need to be devoted to MSE calculation, while the decode costs remain constant, because skipped frames may also be decoded due to the sequential nature of modern video encoding schemes like H.264.

In Figure 17, we vary the decoding speed of our emulated on-disk ASIC (240/480/960 FPS) and compare it to the baseline. We use two sampling rates: 10% and 50%. Higher video decoding capacity in the disk leads to higher application-level throughput, but not proportionally, due to overheads of MSE computation. When sampling rate is high (Figure 17b), application throughput is lower (600 vs. 800), because more MSE computation is needed. Meanwhile, host-disk transfer rate will be higher (800 vs. 300 MB/s), as more decoded frames are sent over the bus. Overall, we observe that decode-enabled storage must have decode capability of over 480 FPS (at 720p) to outperform baseline on this task.

YFCC/Red bus YFCC/Obama VIRAT/Pedestrian

Figure 18: Example Results from Full Visual Pipelines

7.3 Full End-to-End Pipelines

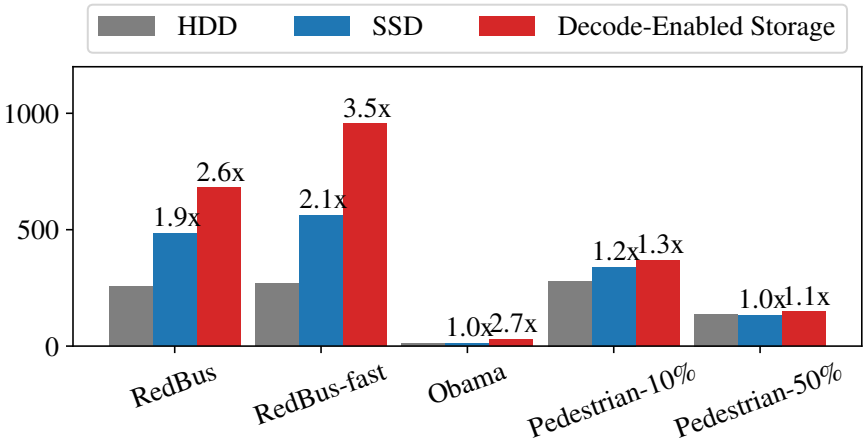
Finally, we evaluate the following full analytics pipelines applicable to real image/video search tasks:

- `RedBus` runs the pipeline in Figure 3 to find red buses in YFCC100M. It first runs a redness color filter, and then passes the candidates to an SSD-MobileNet [54] running on the GPU to detect the presence of buses.
- `RedBus-fast` trades off accuracy for speed by replacing object detection with image classification (MobileNet [55]). Classification is faster, but may miss images where the bus is not the dominant object.
- `Obama` searches for Barack Obama in YFCC100M. It first runs face detection to discard images without faces, and then runs face recognition on the face patches.
- `Pedestrian` detects humans in VIRAT videos. It performs frame sampling and uses image difference to filter sampled frames. Because the VIRAT videos are captured from wide angles and far distances, the candidate frames are passed to Faster R-CNN ResNet101 to detect humans, a more expensive but accurate DNN than SSD-MobileNet for this kind of task. We evaluate this with two frame-sampling rates: 10% and 50%.

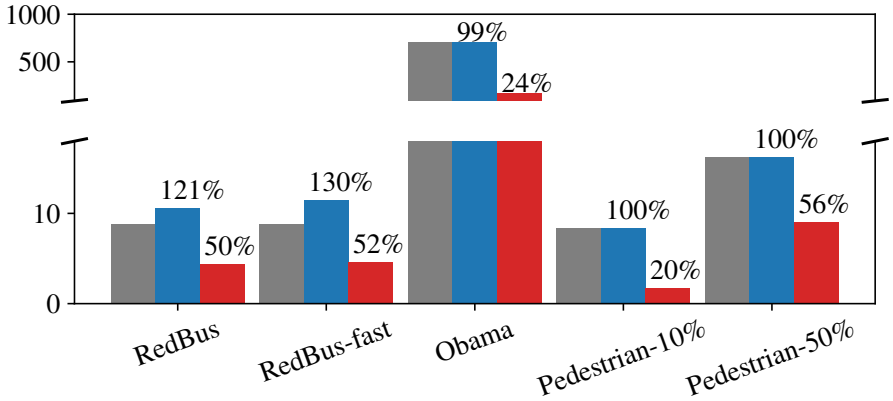
Figure 18 gives a search result example of each application. The selectivity — fraction of images/frames that contain the search target — is 0.01% for `RedBus`, 0.004% for `Obama`, and 2.45% for `Pedestrian`.

We compare the performance of these tasks on (1) a standard SATA HDD; (2) a standard SATA SSD — today's go-to choice for fast but expensive storage; and (3) our proposed NVMe-connected Decode-enabled Storage. Figure 19 reports the application throughput and CPU cost per image for each application and storage type. The annotated numbers are improvements relative to the standard SATA HDD.

Overall, decode-enabled storage shows greater throughput with lower host CPU cost than standard HDD and standard SSD. Comparing `RedBus` and `RedBus-fast`, we observe that this commonly used classification-detection tradeoff is effective only when the disk and decode overhead is removed. With `Obama`, throughput is limited by face detection either in software or hardware. Similarly, `Pedestrian` is largely limited by video decode speed, which is similar in software and hardware. Nonetheless, offloading decode reduces the host CPU cost by 50–80%, allowing more apps to run in parallel on an edge node.



(a) Application Throughput (Images / sec)



(b) CPU Time (Milliseconds) per Image

Figure 19: Full End-to-End Visual Pipeline Performance

8 Related Work

Edge computing is an emerging computing paradigm in which miniature cloud-like compute infrastructure is deployed close to mobile and IoT devices [56]. Prior work on edge computing has focused on network and CPU/GPU resources [57, 58, 17, 59, 60, 61]. They have not investigated storage optimizations for the edge.

Decode-enabled storage complements improvements in other layers of multimedia systems, such as visual data encoding [62, 63, 64], hardware accelerators [33, 34, 35, 36, 46, 47, 48, 49, 50], computer vision algorithms [65, 53, 55, 66], and distributed data processing systems, including some optimized for multimedia data [67, 68, 69, 70, 71, 72].

Embedding application-level processing inside storage, to create *intelligent* or *active disks* [41], has a long and distinguished history. An excellent account of the origin and history of the active disk concept is provided by Riedel [73]. That work attributes the roots of this concept to research on database machines from the mid-1970s through the early 1990s [74, 75, 76, 77, 78, 79, 80]. Riedel's work confirmed the significant performance benefits of this approach for systems of the late 1990s to early 2000s.

Various other researchers have also investigated this concept, including Acharya et al [37], Keeton et al [38], Ma et al [39], Memik et al [40], Rubio et al [42], and Wickremesinghe et al [43]. Closely-related work on optimal function placement at different levels of the memory hierarchy include Abacus [81], Coign [82], River [83], and Eddies [84]. By the mid-2000s, interest in active disks had faded. Their predicted wins were muted by the onward march of commodity hardware performance through Moore's Law. By the late 2000s, active disks appeared to be an idea whose time had come and gone.

There has recently been renewed interest in executing application code close to storage in SSDs [85, 86, 87, 88, 89, 90]. Most of this work focuses on file system and database workloads rather than multimedia processing. For magnetic disks, we are not aware of any published work on active disks in the past 15 years. Our work thus represents a revisit of fundamental questions concerning the optimal placement of processing in a long pipeline that originates on disk.

9 Closing Thoughts

Edge computing faces difficult business challenges as it makes the journey from concept to reality. We will soon see the emergence of small, standalone cloudlets with WiFi or small-cell 5G wireless connectivity that can be easily deployed on premises, at remote outdoor work sites, or on board vehicles, seacraft and aircraft. These standalone cloudlets will enable an explosion of visual data analytics applications. This paper shows how the scalability (and, hence, business viability) of these cloudlets can be greatly improved. Our work provides the storage industry with a unique opportunity to create decode-enabled disks. However, application writers incur little risk in adopting the API proposed in Section 4. If the storage industry fails to capitalize on this opportunity, the competition can step in by providing one of the alternative implementations that were discussed in Section 5. In either case, the bigger point of this paper holds: i.e., *hardware acceleration of visual data decode is valuable for edge computing.*

Our abstraction of decode-enabled storage embodies a new API that consolidates and optimizes common read and decode steps. We have designed, emulated, and validated this new storage abstraction. Our experiments show that use of this abstraction can lower CPU utilization on a cloudlet by up to 50–80%, thereby significantly improving scalability. In addition, we show up to 3.5X improvement in the total elapsed time for processing a typical visual analytics pipeline.

Embedding application-level processing inside storage, to create *intelligent storage* [38] or *active disks* [41] is an old idea. There has recently been renewed interest in this idea for SSDs [85, 86, 87, 88, 89, 90]. Most of that work focuses on file system and database workloads rather than visual data analytics. For magnetic disks, we are not aware of any relevant published work in the past 15 years. Our work thus revisits fundamental questions concerning the optimal placement of processing in a long pipeline that originates on disk.

References

[1] Muhammad Ali et al. “Edge Enhanced Deep Learning System for Large-Scale Video Stream Analytics”. In: *2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC)*. Washington, D.C., 2018.

[2] Ganesh Ananthanarayanan et al. “Real-Time Video Analytics: The Killer App for Edge Computing”. In: *IEEE Computer* 50.10 (2017).

[3] Mahadev Satyanarayanan et al. “Edge Analytics in the Internet of Things”. In: *IEEE Pervasive Computing* 14.2 (2015).

[4] Mahadev Satyanarayanan et al. “Cloudlet-based Just-in-Time Indexing of IoT Video”. In: *Proceedings of the IEEE 2017 Global IoT Summit*. Geneva, Switzerland, 2017.

[5] Simoens, P., Xiao, Y., Pillai, P., Chen, Z., Ha, K., Satyanarayanan, M. “Scalable Crowd-Sourcing of Video from Mobile Devices”. In: *Proceedings of the 11th International Conference on Mobile Systems, Applications, and Services (MobiSys 2013)*. 2013.

[6] Rajesh Balan et al. “The Case for Cyber Foraging”. In: *Proceedings of the 10th ACM SIGOPS European Workshop*. 2002.

[7] Eduardo Cuervo et al. “MAUI: Making Smartphones Last Longer with Code Offload”. In: *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*. 2010.

[8] Mahadev Satyanarayanan. “Pervasive Computing: Vision and Challenges”. In: *IEEE Personal Communications* 8.4 (2001).

[9] MyLio.com. *Here’s How Many Digital Photos Will Be Takein in 2017*. `https://focus.mylio.com/tech-today/how-many-digital-photos-will-be-taken-2017-repost`. 2017.

[10] Jeff Schultz. *How Much Data is Created on the Internet Each Day?* `https://blog.microfocus.com/how-much-data-is-created-on-the-internet-each-day/`. 2019.

[11] Bart Thomee et al. “YFCC100M: the new data in multimedia research”. In: *Communications of the ACM* (2016).

[12] Sangmin Oh et al. “A large-scale benchmark dataset for event recognition in surveillance video”. In: *CVPR 2011*. IEEE. 2011.

[13] Dhruv Parthasarathy. *A Brief History of CNNs in Image Segmentation: From R-CNN to Mask R-CNN*. `https://blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4`. Last accessed on May 17, 2018. 2017.

[14] Daniel Kang et al. “NoScope: Optimizing Neural Network Queries Over Video at Scale”. In: *Proceedings of International Conference on Very Large Data Bases*. 2017.

[15] Daniel Kang, Peter Bailis, and Matei Zaharia. *BlazeIt: Fast Exploratory Video Queries using Neural Networks*. arXiv:1805.01046v1. 2018.

[16] Kevin Hsieh et al. “Focus: Querying large video datasets with low latency and low cost”. In: *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI) 18*. 2018, pp. 269–286.

[17] Christopher Canel et al. “Scaling video analytics on constrained edge nodes”. In: *SysML* (2019).

[18] Larry Huston et al. “Diamond: A Storage Architecture for Early Discard in Interactive Search.” In: *USENIX FAST*. 2004.

[19] Mike Mesnier, Gregory R Ganger, and Erik Riedel. “Object-based storage”. In: *IEEE Communications Magazine* 41.8 (2003).

[20] Seagate. *Kinetic HDD*. `https://www.seagate.com/support/enterprise-servers-storage/nearline-storage/kinetic-hdd/`.

[21] Eric Chung et al. “Single-Chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPGPUs?” In: *Proc. of the 43rd Annual IEEE/ACM Intl. Symp. on Microarchitecture (MICRO-43)*. 2010.

[22] Rehan Hameed et al. “Understanding Sources of Inefficiency in General-Purpose Chips”. In: *Proc. of the 37th Annual Intl. Symp. on Computer Architecture*. 2010.

[23] Dewan Ibtesham. “Improving Large Scale Application Performance via Data Movement Reduction”. PhD thesis. University of New Mexico, 2017.

[24] Peter Jeffcock. *Minimize Data Movement*. `https://blogs.oracle.com/bigdata/minimize-data-movement`. Last accessed January 10, 2021.

[25] Patrick Michael Leyshock. “Optimizing Data Movement in Hybrid Analytic Systems”. PhD thesis. Portland State University, 2014.

[26] *NVM Express*. `https://nvmexpress.org/`.

[27] Chris Mellor. *Will NVMe become the universal block storage access protocol?* `https://blocksandfiles.com/2020/05/27/nvme-universal-block-storage-access-protocol/`. Last accessed: August 23, 2020. 2020.

[28] Billy Tallis. *Western Digital Launches New WD Black NVMe SSDs And Thunderbolt Dock*. `https://www.anandtech.com/show/16149/western-digital-launches-new-wd-black-nvme-ssds-and-thunderbolt-dock`. Last accessed January 21, 2021. 2020.

[29] J. N. Teoh et al. “FPGA implementation of nonlinear control on hard disk drive”. In: *2009 IEEE International Conference on Control and Automation*. 2009.

[30] W. Wu, H. Su, and Q. Wu. “Implementing a Serial ATA Controller Base on FPGA”. In: *Second International Symposium on Computational Intelligence and Design*. 2009.

[31] K. Thongkhome, C. Thanavijitpun, and S. Choomchuay. “A FPGA design of AES core architecture for portable hard disk”. In: *International Joint Conference on Computer Science and Software Engineering (JCSSE)*. 2011.

[32] K. Sengchuai et al. “FPGA-based hardware-in-the-loop verification of dual-stage HDD head position control”. In: *IEEE Regional Symposium on Micro and Nanoelectronics (RSM)*. 2015.

[33] Jahanzeb Ahmad et al. “FPGA Based Implementation of Baseline JPEG Decoder”. In: *Proceedings of the 7th International Conference on Frontiers of Information Technology*. 2009. URL: http://doi.acm.org/10.1145/1838002.1838035.

[34] George Kyrtsakas and Roberto Muscedere. “An FPGA implementation of a custom JPEG image decoder SoC module”. In: *IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*. IEEE. 2017.

[35] Xilinx. *Xilinx JPEG decoder*. https://www.xilinx.com/products/intellectual-property/1-4dcu5s.html. Last accessed: September 12, 2019. 2019.

[36] Intel. *Intel JPEG decoder core*. https://www.intel.com/content/www/us/en/programmable/solutions/partners/partner-profile/a2e-technologies/ip/jpeg-decoder-core.html. Last accessed September 12, 2019. 2019.

[37] A. Acharya, M. Uysal, and J. Saltz. “Active Disks: Programming Model, Algorithms and Evaluation”. In: *Proceedings of Architectural Support for Programming Languages and Operating Systems*. 1998.

[38] K. Keeton, D. Patterson, and J. Hellerstein. “A Case for Intelligent Disks (IDISKs)”. In: *ACM SIG on Management of Data Record* (1998).

[39] X. Ma and A. Reddy. “MVSS: An Active Storage Architecture”. In: *IEEE Transactions On Parallel and Distributed Systems* (2003).

[40] G. Memik, M. Kandemir, and A. Choudhary. “Design and Evaluation of Smart Disk Architecture for DSS Commercial Workloads”. In: *Proceedings of the International Conference on Parallel Processing*. 2000.

[41] E. Riedel, G. Gibson, and C. Faloutsos. “Active Storage for Large-Scale Data Mining and Multimedia”. In: *Proceedings of Very Large Data Bases*. 1998.

[42] J. Rubio, M. Valluri, and L. John. *Improving Transaction Processing using a Hierarchical Computing Server*. Tech. rep. TR-020719-01. Laboratory for Computer Architecture, The University of Texas at Austin, 2002.

[43] R. Wickremisinghe, J. Vitter, and J. Chase. “Distributed Computing with Load-Managed Active Storage”. In: *Proceedings of IEEE International Symposium on High Performance Distributed Computing*. 2002.

[44] Cision PRWeb. *Introducing the Highest Performance and Most Power Efficient 4Kp120 HEVC/H.265 Decoder*. http://www.prweb.com/releases/2014/01/prweb11491436.htm. Last accessed September 17, 2019. 2014.

[45] SOC Technologies. *H.264 4K Video Decoder Chipset*. https://www.soctechnologies.com/chipsets/chipset-h264-4k-decoder. Last accessed September 17, 2019. 2019.

[46] Yuichi Hori and Tadahiro Kuroda. “A 0.79-mm^2 29-mW Real-Time Face Detection Core”. In: *IEEE Journal of solid-state circuits* (2007).

[47] Changjian Gao and Shih-Lien Lu. "Novel FPGA based Haar classifier face detection algorithm acceleration". In: *2008 International Conference on Field Programmable Logic and Applications*. 2008. DOI: 10.1109/FPL.2008.4629966.

[48] Junguk Cho et al. "Fpga-based face detection system using haar classifiers". In: *ACM/SIGDA international symposium on Field programmable gate arrays*. 2009.

[49] Chun He, Alexandros Papakonstantinou, and Deming Chen. "A novel SoC architecture on FPGA for ultra fast face detection". In: *2009 IEEE International Conference on Computer Design*. 2009.

[50] Seunghun Jin et al. "Design and implementation of a pipelined datapath for high-speed face detection using FPGA". In: *IEEE Transactions on Industrial Informatics* (2011).

[51] John S Bucy et al. *The DiskSim simulation environment version 4.0 reference manual*. 2008.

[52] John Linwood Griffin et al. "Timing-accurate storage emulation". In: *Conference on File and Storage Technologies*. 2002.

[53] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of IEEE Computer Vision and Pattern Recognition*. 2016, pp. 770–778.

[54] Wei Liu et al. "SSD: Single shot multibox detector". In: *European Conference on Computer Vision*. 2016, pp. 21–37.

[55] Andrew G Howard et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861* (2017).

[56] Mahadev Satyanarayanan. "The Emergence of Edge Computing". In: *IEEE Computer* 50.1 (2017).

[57] Angela H Jiang et al. "Mainstream: Dynamic stem-sharing for multi-tenant video processing". In: *2018 USENIX Annual Technical Conference*. 2018, pp. 29–42.

[58] Junjue Wang et al. "Towards Scalable Edge-Native Applications". In: *Proceedings of the Fourth IEEE/ACM Symposium on Edge Computing (SEC 2019)*. Washington, DC, 2019.

[59] Shu Shi et al. "Mobile VR on Edge Cloud: A Latency-Driven Design". In: *Proceedings of the 10th ACM Multimedia Systems Conference*. 2019. DOI: 10.1145/3304109.3306217. URL: https://doi.org/10.1145/3304109.3306217.

[60] Bo Hu and Wenjun Hu. "LinkShare: Device-Centric Control for Concurrent and Continuous Mobile-Cloud Interactions". In: *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*. 2019. DOI: 10.1145/3318216.3363303. URL: https://doi.org/10.1145/3318216.3363303.

[61] Jeffrey Helt et al. "Sandpaper: Mitigating Performance Interference in CDN Edge Proxies". In: *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*. 2019. DOI: 10.1145/3318216.3363313. URL: https://doi.org/10.1145/3318216.3363313.

[62] Aaron Koehl and Haining Wang. "SERF: Optimization of Socially Sourced Images Using Psychovisual Enhancements". In: *Proceedings of the 7th International Conference on Multimedia Systems*. 2016. DOI: 10.1145/2910017.2910609. URL: https://doi.org/10.1145/2910017.2910609.

[63] Yanyuan Qin et al. “Quality-Aware Strategies for Optimizing ABR Video Streaming QoE and Reducing Data Usage”. In: *Proceedings of the 10th ACM Multimedia Systems Conference*. 2019. DOI: 10.1145/3304109.3306231. URL: https://doi.org/10.1145/3304109.3306231.

[64] Hui Su et al. “Context-Adaptive Recursive-Filtering-Based Intra Prediction in Video Coding”. In: *Proceedings of the 24th ACM Workshop on Packet Video*. New York, NY, USA, 2019. DOI: 10.1145/3304114.3325615. URL: https://doi.org/10.1145/3304114.3325615.

[65] Shaoqing Ren et al. “Faster R-CNN: Towards real-time object detection with region proposal networks”. In: *Advances in neural information processing systems*. 2015.

[66] Ben Hamlin, Ryan Feng, and Wu-chi Feng. “ISIFT: Extracting Incremental Results from SIFT”. In: *Proceedings of the 9th ACM Multimedia Systems Conference*. 2018. DOI: 10.1145/3204949.3210549. URL: https://doi.org/10.1145/3204949.3210549.

[67] Harsh Agrawal et al. “Cloudcv: Large-scale distributed computer vision as a cloud service”. In: *Mobile cloud visual media computing*. Springer, 2015.

[68] Gylfi Guundefinedmundsson et al. “Towards Engineering a Web-Scale Multimedia Service: A Case Study Using Spark”. In: *Proceedings of the 8th ACM on Multimedia Systems Conference*. 2017. DOI: 10.1145/3083187.3083200. URL: https://doi.org/10.1145/3083187.3083200.

[69] Konstantin Pogorelov et al. “A Holistic Multimedia System for Gastrointestinal Tract Disease Detection”. In: *Proceedings of the 8th ACM on Multimedia Systems Conference*. 2017. DOI: 10.1145/3083187.3083189. URL: https://doi.org/10.1145/3083187.3083189.

[70] Chen Song et al. “Scalable Distributed Visual Computing for Line-Rate Video Streams”. In: *Proceedings of the 9th ACM Multimedia Systems Conference*. New York, NY, USA, 2018. DOI: 10.1145/3204949.3204974. URL: https://doi.org/10.1145/3204949.3204974.

[71] Miao Zhang et al. “Video Processing with Serverless Computing: A Measurement Study”. In: *ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. 2019. DOI: 10.1145/3304112.3325608. URL: https://doi.org/10.1145/3304112.3325608.

[72] Zhou Fang, Dezhi Hong, and Rajesh K. Gupta. “Serving Deep Neural Networks at the Cloud Edge for Vision Applications on Mobile Platforms”. In: *Proceedings of the 10th ACM Multimedia Systems Conference*. 2019. DOI: 10.1145/3304109.3306221. URL: https://doi.org/10.1145/3304109.3306221.

[73] Erik Riedel. “Active Disks — Remote Execution for Network-Attached Storage”. CMU-CS-99-177. PhD thesis. Department of Electrical and Computer Engineering, Carnegie Mellon University, 1999.

[74] H. Boral and D.J. DeWitt. “Database Machines: An Idea Whose Time Has Passed?” In: *International Workshop on Database Machines*. 1983.

[75] D.K. Hsiao. “DataBase Machines Are Coming, DataBase Machines Are Coming!” In: *IEEE Computer* 12.3 (1979).

[76] D.J. DeWitt. “DIRECT - A Multiprocessor Organization for Supporting Relational Database Management Systems”. In: *IEEE Transactions on Computers* 28.6 (1979).

[77] D.J. DeWitt and P.B. Hawthorn. “A Performance Evaluation of Database Machine Architectures”. In: *Proceedings of VLDB*. 1981.

[78] D. J. Dewitt et al. “The Gamma Database Machine Project”. In: *IEEE Transactions on Knowledge and Data Engineering* 2.1 (1990).

[79] G. Qadah and K. B. Irani. “A Database Machine for Very Large Relational Databases”. In: *IEEE Transactions on Computers* C-34.11 (1985).

[80] S.Y.W. Su et al. “The Architectural Features and Implementation Techniques of the Multicell CASSM”. In: *IEEE Transactions on Computers* 28.6 (1979).

[81] K. Amiri et al. “Dynamic Function Placement for Data-Intensive Cluster Computing”. In: *USENIX Annual Technical Conference*. 2000.

[82] G. Hunt and M. Scott. “The Coign Automatic Distributed Partitioning System”. In: *Proceedings of USENIX Operating Systems Design and Implementation*. 1999.

[83] R. Arpaci-Dusseau et al. “Cluster I/O with River: Making the Fast Case Common”. In: *Proc. Input/Output for Parallel and Distributed Systems*. 1999.

[84] R. Avnur and J. Hellerstein. “Eddies: Continuously Adaptive Query Processing”. In: *ACM SIGMOD*. 2000.

[85] Simona Boboila et al. “Active Flash: Out-of-core Data Analytics on Flash Storage”. In: *Proc. of the 28th IEEE Mass Storage Symposium*. 2012.

[86] Devesh Tiwari et al. “Active Flash: Towards Energy-Efficient, In-Situ Data Analytics on Extreme-Scale Machines”. In: *Proceedings of the File and Storage Technologies Conference*. 2013.

[87] Yangwook Kang et al. “Enabling cost-effective data processing with smart ssd”. In: *2013 IEEE 29th symposium on mass storage systems and technologies (MSST)*. 2013.

[88] Jaeyoung Do et al. “Query processing on smart SSDs: opportunities and challenges”. In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. 2013.

[89] Benjamin Y Cho et al. “XSD: Accelerating mapreduce by harnessing the GPU inside an SSD”. In: *Proceedings of the 1st Workshop on Near-Data Processing*. 2013.

[90] Jianguo Wang et al. “SSD in-storage computing for list intersection”. In: *Proceedings of the 12th International Workshop on Data Management on New Hardware*. 2016.