Adaptive Wide-area Streaming Analytics for the Internet of Things

Paper #97, 12 pages

ABSTRACT

In this paper, we present AdaptiveStream, a stream processing system for real-time analytical applications in the wide area. To cope with the scarce and varying available bandwidth, AdaptiveStream allows applications to trade-off accuracy for data freshness. To achieve this, AdaptiveStream first performs an offline profiling that explores different degradation strategies and generates an accuracy-bandwidth profile. During runtime, AdaptiveStream adapts the application to match the rate of measured bandwidth.

We evaluate AdaptiveStream against three real-world applications including surveillance applications, augmented reality and distributed log analysis. At places where traditional approaches would lead to either significant application accuracy drop or long tail latency, our system gracefully maintains the balance between application accuracy and system performance.

1. INTRODUCTION

Wide-area streaming analytics are becoming pervasive, especially with the emerging Internet-of-Thing (IoT) applications. Large cities such as New York, Beijing and Seattle are deploying millions of cameras for traffic control [2, 3]. Retails stores and critical areas such as railway stations are also being monitored for abnormal activities. Buildings are increasingly equiped with a wide variety of sensors to improve building energy use, occupant comfort, reliability and maintenance [25]. Geo-distributed infrastructure, such as Content Delivery Network (CDN), needs to analyze user requests (machine logs) and optimize data placement to improve delivery efficiency. In these problems, the data collected at the edge needs to be transported acoross the wide-area and analyzed in near real-time.

Existing stream processing for "big data", such as Borealis [4], Storm [44], or Spark Streaming [48], only work in the context of a single cluster, where the bandwidth is sufficient (or at least easier to provision). While they are the perfect backends for analyzing streams once data arrive, in wide-area, the network, with scarce and variable bandwidth, easily becomes the bottleneck [37]. What's worse, the growth rate of wide-area network capacity is not keeping up with the increasing rate of traffic [23].

When facing situations where the bandwidth is not sufficient, applications deployed today either choose a conservative setting (e.g. only delivering 360p videos) or leave their fate to the underlying transport layer: (1) in the case of TCP, the sender will be blocked and data are backlogged, leading to severe delay; (2) in the case of UDP, uncontrolled packet loss occurs, leading to application performance drop.

The idea of adaptive streaming and trading accuracy for data freshness is not new. Multimedia applications [30, 42] has been extensively studied in the past. However, their optimization goal is to improve user experience; resulting in an adaptation strategy with a mismatch from our target video analytical applications (§2.1).

In this paper, we present the design and implementation of AdaptiveStream, a stream processing systems for the wide-area. Our system enables applications making explicit trade-off between data fidelity and freshness with empirical performance models.

Our system allows developers construct applications that uses the computing capabilities at the data source site for pre-processing, which has been demonstrated effective in reducing the bandwidth demand [36, 46]. There are two types of pre-procssing operations: lossless or lossy. While lossless transformations are helpful [37], lossy operations often allow more savings at the expense of reduced accuracy. In this paper, we primarily study the effect of lossy transformations, i.e. degradation.

In video applications, reducing image resolution, frame rate or changing the video encoding quality are potential degradation operations that affects data rate and application accuracy. In log analysis, a local thresholding before transmitting the data is an example of degradation. We notice that these operations are often more than binary decision; rather they are tunable *knobs*.

Although the basic idea behind degradation is intuitive, realizing them in practice is non-trivial. First, degradation has different impact for different application and data. Second, each degradation is often more than a binary decision; they are often parameterized and has different impact. Real-world application also have multiple knobs to tune. It's not always possible to derive a closed form of analytical relationship between the degradation and its impact on bandwidth/accuracy.

We elaborate on the challenges in §3.1.

To make degradation and adaptation practical, AdaptiveStream employs a data-driven empirical-analysis approach. First of all, developers express degradation operations with AdaptiveStream APIs. These operations are merely hints rather than exact rules as we do not assume developers are well-aware of the degradation impact (§4.1). Instead, AdaptiveStream performs an automatic multi-dimensional profiling that produces the bandwidth-accuracy trade-off *profile*. Developers do need to provide a representative dataset and a utility function to measure the degradation impact (§4.2). The profile is used in our runtime system. Aided with bandwidth estimation and congestion controller, the runtime ensures an adaptive execution in the case of network variation (§4.3).

To study the effect of degradation, we've built three real-world applications using AdaptiveStream: a street surveillance application performing pedestrian detection, an augmented reality application recognizing everyday objects and a distributed top-k application analyzing web server access logs. The first two video streaming applications have three knobs: resolution, frame rate and video encoding quality. For the distributed top-k application, we have two knobs: N in the local top-N operation and T as a local threshold.

Our offline profiling stage explores the design space for all three applications and generates the profile consists of Pareto-effiecient configurations. Our results show that degradation operations often have different impact and many optimal configurations are only possible when they are combined.

We also evaluate our system's runtime behavior against baseline with controlled experiment. The evaluations shows that AdaptiveStream can gracefully handle network deterioration where a bounded latency is achieved without too much application accuracy drop. Even when the bandwidth is nearly halved, we can still achieve 80% accuracy with 15 seconds delay where TCP creates backlogged for minutes and UDP only offers accuracy less than 50%.

In summary, this paper makes the following contributions:

- We study in depth of wide-area streaming applications in the case of network resource variation.
- We design APIs to allow for exploring bandwidth-accuracy trade-off with minimal developer effort.
- We implement the system to perform application profiling and runtime adaptation.
- We build three real-world applications and evaluate their behavior under different scenarios.

2. MOTIVATION

In this section, we make the case for an adaptive stream processing system in the wide area by examining the gap between application demands and the existing infrastructure. We start with a few streaming applications.

Video Surveillance: We envisage a city-wide monitoring system that aggregates camera feeds (both stationary ground cameras and moving aerial vehicles) and analyzes video streams in real-time for surveillance, anomaly detection or business intelligence [33]. While traditionally human labors are involved in analyzing abnormal activities, recent advances in computer vision and deep learning has dramatically increased the accuracy for automatic analysis of visual scenes, such as pedestrian detection [19], vehicle tracking [16], or facial recognition to locate people of interest [35, 28].

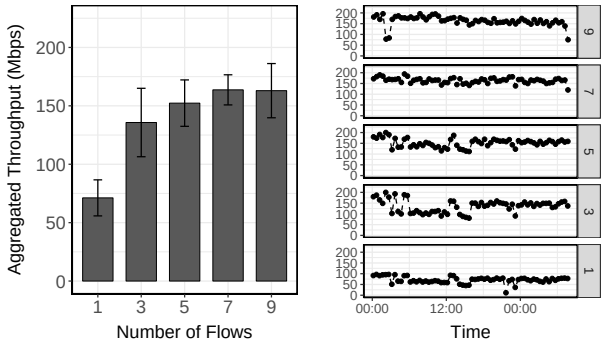


Figure 1: Bandwidth measurement between Amazon EC2 sites (from Ireland to California). Note the time-series plot has a resolution of 30 minutes.

IoT Sensors: While traditional environmental sensors are slow [9], we are seeing an increasing trend with high-frequency, high-precision sensors being deployed. For example, the uPMU monitoring system for the electrical grid consists of a network of 1000 devices; each produces 12 streams of 120 Hz high-precision values accurate to 100 ns. This amounts to 1.4 million points per second that requires specialized timeseries database [8].

Log Analysis: Large organizations today are managing 10–100s of datacenters (DCs) and edge clusters worldwide [13]. While most log analysis today runs in a batch mode and on a daily basis, there is trend in analyzing logs in real-time for quicker optimization. For example, a content distribution network (CDN) can improve the overall efficiency by optimizing data placement if the access logs can be processed in real-time.

We consider the practical issues with deploying these applications in the wide-area. Our stand is that these applications face a bigger network challenge. Data generated from the edge often fail to be delivered to the the processing site because of the scarce and variable bandwidth capacity in the wide-area. Once they arrive, existing stream processing systems can easily manage a large cluster and perform data analytics at real-time.

2.1 Wide-Area Bandwidth Characteristics

To understand the bandwidth characteristics in the wide-area, we conducted a simple measurement using Amazon EC2. We use iPerf [1] to measure the pair-wise bandwidth between four geo-distributed sites throughout the day. We

observed large variance in the measured bandwidth and one such pair of sites is shown in Fig. 1. Regardless of the number of flows[1], there exist occasions when the available bandwidth is almost halved. We believe the back-haul links between EC2 sites are better (if not at least representative) in comparison to the general wide-area links. The varying nature poses real challenge to the realization and successful deployment of wide-area streaming applications.

2.2 Making the Case for a System Approach

When facing insufficient resources, applications that do not adapt will suffer severe performance penalty: such as a backlog of data for TCP or uncontrolled packet loss for UDP. We argue that applications should be made adaptive to the varying resource.

While adaptive streaming exist in certain application domain, there has not been a general solution. Consider video streaming applications that have been extensively studied in the literature. There are a plethora of encoding techniques [39, 21] with adaptive strategies [47, 30, 34], however, their primary goal is to optimize end-user quality of experience (QoE). When end users consume a video clip, a smooth video is often more enjoyable than videos with intermittent pauses, even though with crisp images.

Video streaming analytics often have different goals, therefore entail different adaptive strategies. For example, many computer vision detection algorithms rely on the edge information [14, 27, 45] while object tracking applications works best when the inter-frame difference is small [7].

Further, even the same query, when used in different context, requires different strategies. For example, pedestrian detection can be deployed with a ground stationary camera. When taking pedestrian walking speed into consideration, it's not necessary to guarantee a high frame rate. But because these surveillance cameras are far from the target, it's crucial to have a high-resolution and sharp image. On the other hand, when deployed with a mobile phone, due to the movement of the camera, reducing frame rate will introduce significant errors. Fig. 2 illustrates the different characteristics.

This motivates us to take a system-level approach that synthesize different adaptation strategies for different queries and contexts.

3. AdaptiveStream OVERVIEW

In this section, we present an overview of AdaptiveStream. The primary goal of AdaptiveStream is to enable applications with the ability to adapt its communication in a guided manner.

3.1 Challenges

There are four challenges in realizing AdaptiveStream.

C1: Diverse application and data: As discussed in §2.2, the best adaptation scheme is often application- and context-specific optimizations. It becomes important to separate individual application logic from specific degradation strategy as well as adaptation mechanism.

[1]EC2 has a per-flow and per-VM rate limiting [50].

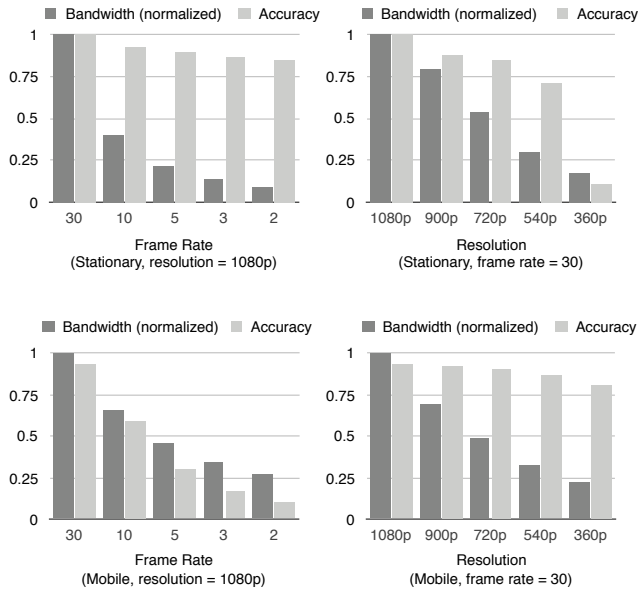


Figure 2: Horizontally, different dimension affects the degradation characteristics differently. Vertically, the same dimension have different impact for different application context.

C2: No analytical solutions: Unlike SQL queries whose demand and accuracy can typically be estimated using analytical models [17], many of our streaming applications are dealing with unstructure data using either use blackbox operations (such as H.264 encoding) or non-linear operators (such as thresholding). The impact of these degradations is not immediately available.

C3: Multi-dimensional exploration: Real-world applications typically have more than one tunable parameters; leading to a combinatorial space for exploration. In addition, these parameters are not necessarily orthognal. The optimal degradation strategies may only be achievable when more than one degradation is in effect.

C4: Runtime adaptation at application layer: Although recent work on resource reservation makes it possible to guarantee quality of service with new IP or MAC layer protocols in LAN (e.g. TSN [24]), we target at WAN analytics where most of the infrastructure is owned by others and shared among many users. An application-layer solution is in favor to those that require special hardware or software upgrade.

3.2 System Architecture

To address the aforementioned challenges, AdaptiveStream's solution is split into four parts (Fig. 3).

Programming abstraction: Applications are modelled as a directed acyclic graph (DAG) of computation and we propose a novel set of `maybe` operators to express the specification

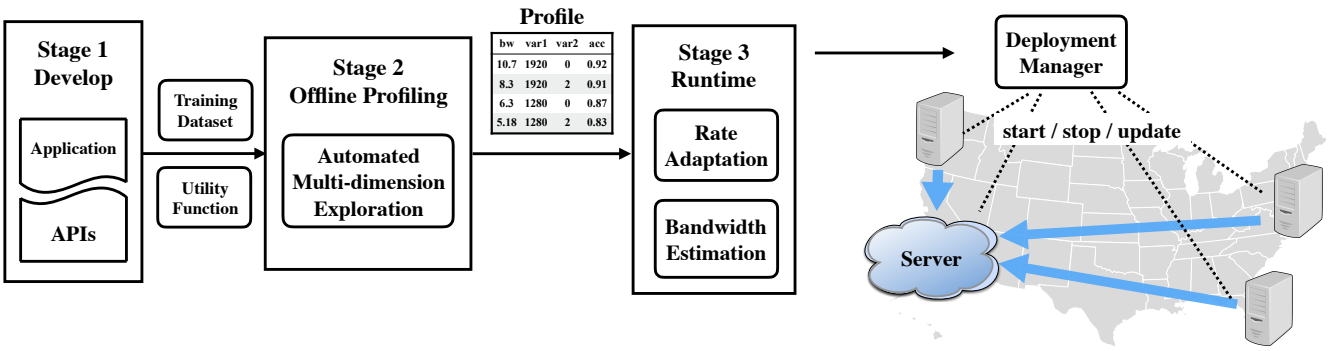


Figure 3: AdaptiveStream Overview.

of degradation. Our propose APIs do not require developers to be exact on the quantity; integrating this into existing applications requires minimal effort (§4.1).

Automatic multi-dimensional profiling: Our system automatically explores the parameter space to generate a Pareto-optimal degradation strategy with a multi-dimensional exploration. We also report our experience and findings about degradation operations (§4.2).

Runtime Adaptation: Finally the streaming application is deployed with a wide-area orchestration manager. At runtime, our system generates additional modules that acts as the control plane to adapt the application execution. The control plane performs bandwidth estimation, congestion monitoring and adaptation. It uses the profile generated from the second stage and adjust the level of degradation (§4.3).

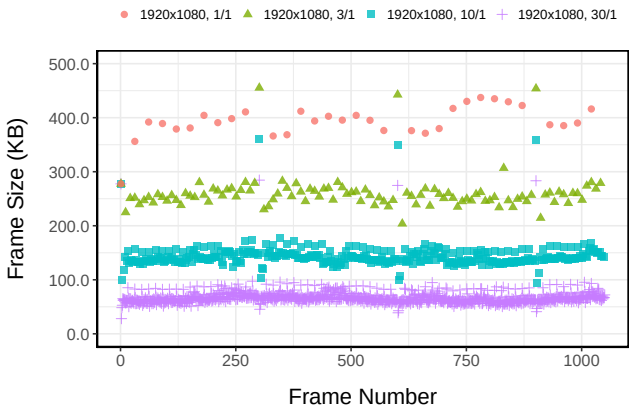


Figure 4: H.264 requires more information per frame when the frame rate is reduced.

4. SYSTEM DESIGN

In this section, we dive into each component of AdaptiveStream; we focus on degradation and adaptation, rather than being verbose on every component.

Applications in AdaptiveStream are composed by connecting a set of operators to form a dataflow processing graph. Our system provides basic set of APIs such as `map`, `filter`, `window` (Table 1). The normal operators are similar to existing stream processing systems; our contribution is the set of APIs to allow the specification of degradation operations for bandwidth-accuracy trade-off.

4.1 Degradation Operators

We first consider a strawman solution: manual policies for degradation. JetStream [37] offers an example: "if bandwidth is insufficient, switch to sending images at 75% fidelity, then 50% if there still isn't enough bandwidth. Beyond that point, reduce the frame rate, but keep the images at 50% fidelity." We identify the following issues with this approach.

Lack of precision: These policies are often developer heuristics and rarely backed up by measurements. First, there is no direct association of the application accuracy with the 75% fidelity configuration. Besides, their impact on the data size is also not trivial. While it seems intuitive that the level of degradation will affect the data size, the precise impact is not always straightforward. For example, one might think that reducing the frame rate by 50% will half the data rate. When video encoding is employed, the inter-frame difference will increased (P-frame size) when the frame rate is reduced. This leads to a larger data size for each frame. Fig. 4 illustrates this complex relationship with an example of H.264 encoding under four different frame rates.

Not scalable: The strawman solution quickly leads to too many policies when multiple degradation operations are involved or a fine-grained control is desired. This manual process becomes tedious and error-prone. When too few rules are provided, the application may oscillate between two rules: one that's too aggressive (always faces insufficient bandwidth) and one that's too conservative.

When using the above strawman solution, developers are forced to manually study and measure the impact of individual degradation policy, prohibiting its wide adoption in practice.

On the other extreme of the design spectrum, a completely developer-free solution is not practical. While static analy-

Normal Operators	*map*(f: I ⇒ O)	Stream<I> ⇒ Stream<O>
	filter(f: I ⇒ bool)	Stream<I> ⇒ Stream<I>
	skip(i: Integer)	Stream<I> ⇒ Stream<I>
	sliding_window(count: Integer, f: Vec<I> ⇒ O)	Stream<I> ⇒ Stream<O>
	tumbling_window(count: Integer, f: Vec<I> ⇒ O)	Stream<I> ⇒ Stream<O>
	timed_window(time: Duration, f: Vec<I> ⇒ O)	Stream<I> ⇒ Stream<O>
	...	...
Degradation Operators	*maybe*(knobs: Vec<T>, f: (T, I) ⇒ I)	Stream<I> ⇒ Stream<I>
	maybe_skip(knobs: Vec<T>)	Stream<I> ⇒ Stream<I>
	maybe_downsample(knobs: Vec<(Integer, Interger)>)	Stream<Image> ⇒ Stream<Image>
	...	...

Table 1: A comparison between normal stream processing operators and our degradation operators. Vec<T> represents a list of elements of type T. Option<T> indicates an element of type T can be absent. Notice the type constrain on the `maybe` function while `maybe_map` relaxes it.

sis has been shown to optimize application execution adaptively in a certain context [15], they do not work well in our dataflow programming model. Static analysis is prone to false positives: exploring wrong or unnecessary parameters. For example, when the application is configured to generates statistics with a `timed_window` operation, static analysis may falsely detect the duration parameter and alter the behavior of the application in an unexpected way. Also, as we will illustrate in §4.2, with each introduced parameter, the profiling time increases drastically as all parameters pose a combinatorial space.

Our system take a middle ground between these two extremes: developers use a novel `maybe` API to annotate degradation operations without being exact on the values. Think of these APIs as hints from developers: this operation, when in use, will likely reduce the data size and affect the data fidelity; however the exact impact is not clear.

The basic form of `maybe` operator takes two arguments: a knob and a degradation function (see Table 1). The knob indicates different degradation levels; the function performs the actual degradation operation with a configurable level. We restrict the type signature of the function that this API can accept: $f(T, I) \Rightarrow I$. That is, the degradation function should not alter the type of the stream. While this might seem a strong restriction, when combined with `map` operator, the system is still expressive enough. We describe our implementation and usage in §5.

Based on the `maybe` primitive, one can implement wrappers for common degradation operations. For example, `maybe_skip` will optionally subsample a stream; and `maybe_downsample` can adjust the image resolution to a configured target. With this API, the example mentioned earlier can now be implemented as follows:

```
let app = Camera::new((1920, 1080, 30))
  .maybe_downsample(vec![(1600, 900), (1280, 720)])
  .maybe_skip(vec![2, 5])
  .map(|frame| frame.show())
  .compose();
```

This snippet first instantiate a `Camera` source, which is a `Stream<Image>` with 1920x1080 resolution and 30 FPS. Two degradation operations are chained after the source: one that downsample the resolution to either 1600x900 or 1280x720; the other skip the frame with a parameter of 2 or 5, resulting in 30/(2+1)=10 FPS or 30/(5+1)= 6 FPS. After the degradation, images are shown on the display. In practice, further processing operators can be chained.

While the API itself has simplified the specification of degradation, the exact amount has to be known for precise rate adjustment at runtime. We then turn to the second stage of our system that performs automatic profiling.

4.2 Multi-dimensional Profiling

The goal of our profiling stage is to explore the bandwidth-accuracy trade-off and generate a *profile* that is Pareto-optimal.

We first define terms and notations we will use. Each `maybe` operator within an application corresponds to a knob k. Suppose the application has n knobs, their combination forms a configuration $c = [k_1, k_2, ...k_n]$. The set of all configurations $\mathbb{C}$ is the space that our profiling system need to explore.

There are two mappings that we are particularly interested: a mapping from a particular configuration to its bandwidth requirement $B(c)$ and the accuracy measure $A(c)$. The Pareto-optimal set $\mathbb{P}$ can then be defined (Eq. 1): for all $c \in \mathbb{P}$, there is no alternative configuration c' that requires less bandwidth while giving a higher accuracy.

$$\mathbb{P} = \{c \in \mathbb{C} : \{c' \in \mathbb{C} : B(c') < B(c), A(c') > A(c)\} = \varnothing\} \quad (1)$$

Since there is often no closed form relation for $B(c)$ and $A(c)$, our system takes a data-driven approach: with a representative dataset and an application-specific utility function, our system evaluates each configuration for their bandwidth demand and accuracy degradation. The degradation could either be measured against the groundtruth; or in the case when labelled dataset is not available, the system uses the reference results when all degradations are turned off.

When only one knob is involved, we can represent its

Symbol	Description
n	number of degradation operations
k_i	the i-th degradation knob
$c = [k_1, k_2, ...k_n]$	one specific configuration
$\mathbb{C}$	the set of all configurations
$B(c)$	bandwidth requirement for c
$A(c)$	accuracy measure for c
$\mathbb{P}$	Pareto efficienct set

Table 2: Notations used in profiling.

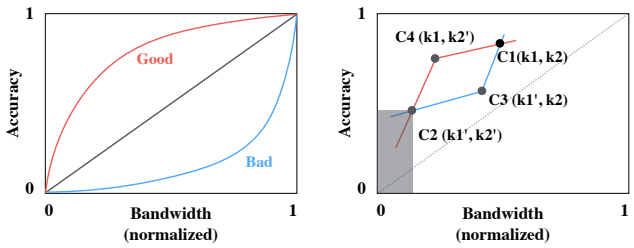


Figure 5: Illustration on the behavior of different degradation operations.

impact using bandwidth-accuracy curve (Fig. 5 left) and quantify the effect with the area under the curve (AUC). Each curve consists of the point with coordinate $(B(c), A(c))$ when configuration c is used. The straight line from (0, 0) to (1, 1) splits the space into two parts. If a degradation's profile lies above this line, it indicates that this operation can be effective for the goal of saving bandwidth while minimizing accuracy drop. For degradation operations that's below the straight line, much of the data information is gone even at minor bandwidth drop. The Pareto-optimal strategy is the profile that has a maximal AUC.

While $B(c)$ and $A(c)$ can be monotonic along individual dimension. When combined, it poses complex behavior. The right half illustrates Fig. 5 illustrates it.

While the complexity prohibits further analytical analysis of the degradation behavior, in practice, several techniques can make the problem tractable. First, Exploring these configurations is an embarrassingly parallel task. The profiling can be made trivially parallel. Besides, when the degradation level increases or when multiple degradation operations are in effect, the amount of data as well as the computational complexity are also dramatically reduced. Moreover, developers can specify a lower bound on the accuracy and the profiling can stop searching the space that is worst than a known configuration (such as the gray area in Fig. 5).

4.3 Runtime Adaptation

At runtime, the user program is automatically converted to a client half and server half; and AdaptiveStream abstracts the communication as well as rate adaptation. Fig. 6 shows our runtime architecture.

Object-level queue: The queue bridges data generation and

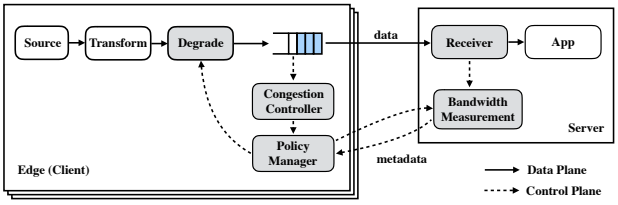


Figure 6: Runtime adaptation system architecture. The grey components are what AdaptiveStream provides.

the network capacity. It transmits data as fast as the network can handle and detects congestion when the queue grows.

Congestion controller: Our first prototype used the queue length as a signal for congestions, this turns out to be a bad match as the degradation operation may change object arrival rate (e.g. lower frame rate for video). Our next version uses data size, but it also leads to long latency when the degradation changes individual object size (e.g. lower resolution). Our current design uses an adaptive size: given current configuration and the profile, the queue learns the desired data generation rate (bps), and with a user configured latency, the queue derive the congestion watermark.

Bandwidth Measurement: The receiver delivers the received data to application. In the meantime, it measures the effective throughput between each client-server pair as an indication of current bandwidth [1]. To avoid spikes in the bandwidth measurement, exponential smoothing is employed. While the receiver performs bandwidth measurement every second, it does not send the information back to each client. avoid unnecessary communication, the client requests the measurement only when congestion is detected.

Policy Manager: Upon receiving signals from congestion controller, it performs an RPC request to the server for current bandwidth measurement. With the learned profile, it determines the degradation level and start the actual degradation strategy. The bandwidth specification in the learned profile is the required bandwidth, the policy manager usually takes a conservative approach: using a constant rate ALPHA to adjust the available bandwidth. When the congestion is resolved, the policy manager gradually reduce the degradation level (additive increase phase).

Degrade: The actual degradation operation is rather simple. Operators based on the `maybe` API supports a `set` function that would change the internals of the operator. The `set` function is invoked when degradation is needed.

5. IMPLEMENTATION

In this section we present details about our implementation, including a prototype framework and three non-trivial wide-area streaming applications. AdaptiveStream is implemented in Rust and open-source on Github.[2]

[2]Url elided for anonymity.

5.1 Framework

While our proposed APIs are general and not language specific, we chose a safe language, Rust, for the core framework for the following reasons. First, Rust's memory safety guarantee can ensure applications running continously for an extended period of time. Besides, the zero-cost abstraction removes the possibilities of tail latencies caused by uncoordinated garbage collection [29]. In addition, we rely on Rust's type system to enforce the type match on `maybe` operations.

All operators implement the `Stream` trait which has an associate type `Item` and a core function `next` that returns `Datum`. Each datum is either an item with the `Stream::Item` or an `Error` that the operator use to communicate with the runtime scheduler. The concrete form of `maybe` API is almost an direct translation of the API specification. While our API specification in Table 1 uses a vector for knobs, our Rust implementation is more general: any type (including vector) that implements `IntoKnob` trait can be used as the knob.

```
pub trait Stream {
   type Item;
   fn next(&mut self) -> Datum<Self::Item, Error>;

   fn maybe<K, F>(self, opts: K, f: F) -> Maybe<Self, F>
      where Self: Sized,
            K: IntoKnob,
            F: FnMut(K::Item, Self::Item) -> Self::Item {

       // omitted
   }
}

pub trait IntoKnob {
   fn into_knob(self) -> Knob;
}
```

Developers can directly use the above API with user-defined functions. The snippet below shows how a quantization degradation can be implemented with our API. First, a vector of values is converted into a stream object. Then a `maybe` operator with a knob value (2 or 3) and a function that performs an interger division (quantization). Function `collect` will run this stream and hold the output in a vector. Depending on the degradation level, the output stream could either be [1, 2, 3, 4], [0, 1, 1, 2], or [0, 0, 1, 1].

```
let quantized_stream = vec![1, 2, 3, 4]
   .into_stream()
   .maybe(vec![2, 3],
          |knob, p| p / knob);
   .collect();
```

We've also extended the basic API for common operations. As we are building video processing applications, we implemented a specialized `maybe_downsample` operator can that wraps `downsample` function internally.

```
fn downsample(res: (usize, usize), image: Mat) -> Mat {

   // omitted

}
```

Applications built with AdaptiveStream runs as a single process. The entire processing pipeline is often specified in a single main file. The execution mode (profiling, runtime as client or runtime as server) is configured with command line arguments or environment variables. Our deployment manager is currently a shell script using Docker container.

5.2 Building AdaptiveStream Applications

Using AdaptiveStream, we've built three applications: pedestrian detection surveillance, an augmented reality and a distributed Top-k (Fig. 7). Table 3 summarizes the application specific part: knobs, utility function and dataset

Pedestrian Detection: This application analyzes video streams from installed CCTV cameras and detect pedestrians inside. The detection result is a list of bounding boxes representing pedestrian's relative location within the view. Variant of this application can be used for safety monitoring, anomaly detection or waiting line counting.

We implement most image-related operations with OpenCV 3.1 [12]. Pedestrians are detected using histogram of oriented gradients (HOG) [18] with the default linear SVM classifier. To ensure real-time processing of frames, GPU-accelerated implementation is used in favor of the CPU-based implementation.

For video encoding, H.264 scheme is chosen for its prevalence in existing systems. Our implemenation is based on GStreamer [43], using `x264enc` plugin. To integrate with AdaptiveStream, we first create a pipeline that exposes `appsrc` (to feed raw image data) and `appsink` (to get encoded bytes). The GStreamer main loop is managed in a separate thread and AdaptiveStream communicates with it via Rust's channel. The `x264enc` is configured with `zerolatency` present and runs using four threads. It uses constant quality encoding and the quantizer is exported as a parameter that can be tuned.

This application has three degradation operations: reducing image resolution, dropping frame rate or lower video encoding quality.

The detection returns a list of bounding boxes; it's compared against a reference result (either the groundtruth or the one without degradation). A successful detection is defined when the intersection over union (IOU) is greater than 50% [20]. For the utility function, we use F1 score (%), the harmonic mean of precision and recall [40].

Augmented Reality: We target at mobile augmented reality applications which offload the heavy computation to resources elsewhere. Although local computation is gaining attraction [41, 49], wireless communication link is also susceptible to capacity variation.

We use a similar setup as the pedestrian detection application except the actual function that analyzes the stream. To recognize objects, we use a a pre-trained neural network [38] that's trained with Imagenet [26]. Similar to our first application, GPU-accelerated implementation is use in favor for real-time processing.

The utility function here is more strict than the pedestrian detection: true-positive depends not only on IOU criteria, but

Application	Knobs	Utility	Dataset
Pedestrian Detection	resolution, framerate, quantizer	F1 score	MOT16-04 (training), MOT16-03 (testing)
Augmented Reality	resolution, framerate, quantizer	F1 score	Video clips of office (training), home (testing)
Top-K	head (N), local threshold (T)	Kendall's W	sec.gov access log (4 days training, 12 days testing)

Table 3: AdaptiveStream Applications

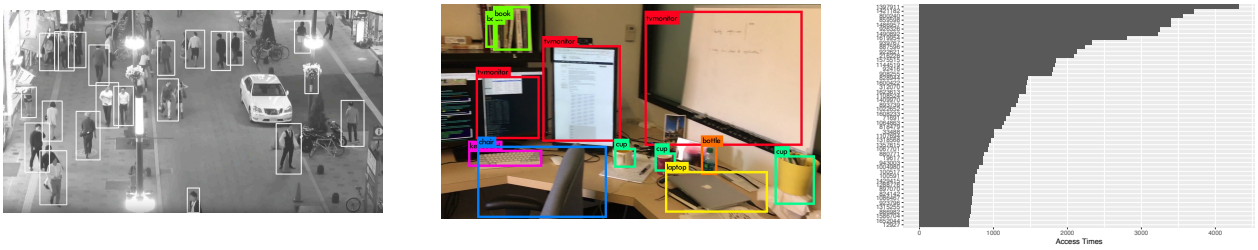


Figure 7: AdaptiveStream applications

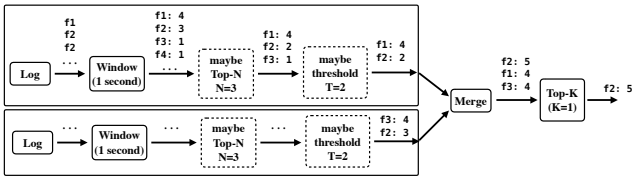


Figure 8: A distributed Top-K application has two tunable parameters: a local Top-N (N) and a local threshold (T).

also on the type of objects (a correct identification).

Distributed Top-K: Many distributed system monitoring applications require to answer the "top-k" question [10], such as the top-k most popular URLs or the top-k most access files. Naive methods of transmitting all the raw log entries to the aggregation point is not feasible as popular servers typically have millions of requests per second. Local worker node can first perform a window-based transformation that generates data summary, such as key-value pairs of `<item, count>`. However, even after this operation, the data size could still be too large given most real-world access patterns follow a long-tailed distribution. There is a large-but-irrelevant tail that is unnecessary to send.

We consider two degradation operations that individual worker nodes can perform: (1) a local Top-`N` operation that shortens the list first; (2) a local threshold `T` that further filters small entries. Obviously, these two operations are not orthognal to each other. Their impact on data size reduction and quality degradation depends on the distribution of the actual data.

we use Kendall's W as the utility function here. It is a distance measure of the concordance between two ranked list. The function outputs a statistic measure ranging from 0 to 1, representing no agreement to complete aggrement, respectively [5].

6. EVALUATION

We first show the generated profile for the three applications. Then for each application, We show how they adapt the behavior at runtime. Under a controlled experiment, even with only transient network capacity drop, our system is able to maintain an end-to-end delay for 10 seconds in the wide-area and accuracy level above 80%. Application-agnostic protocols creates significant backlogged data (TCP for about 100 seconds) or unusable accuracy (UDP).

6.1 Degradation Performance

We describe the dataset we used for offline profiling and interpret the profiling results (Fig. 9) in turn.

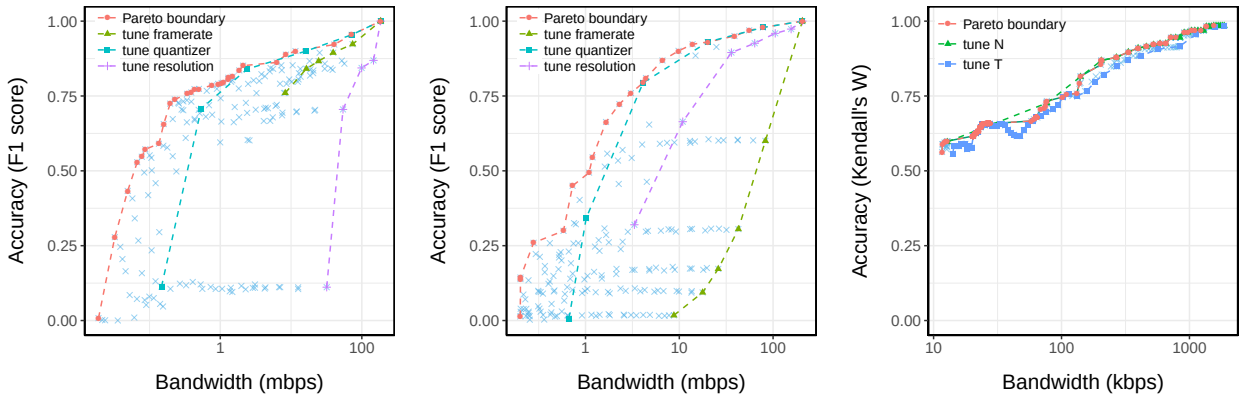
Pedestrian Detection: We use MOT16 dataset [31] to evaluate this application. Specifically we used MOT16-04 as the training dataset. The video feeds capture a busy pedestrian street at night with an elevated viewpoint. The original resolution is 1920x1080, with frame rate 30. The training data has 1050 frames in total, amounting 35-second monitoring. On averge there are 45.3 people per frame.

There are three knobs in this application: resolution, frame rate and encoding quality. To maintain the same 16:9 aspect ratio with the original 1920x1080 resolution, the first degradation only chooses common 16:9 resolutions: 1600x900, 1280x720, 960x540, 640x320. For the framerate, integer values are chosen in favor of fraction values. The original frame rate is 30, and our degradation explores 10, 5, 3, 2, 1. H.264 encoding quantizer has a range from 0 (lossless) to 51 (worst possible), and 18 is the visually lossless [11]. In our experiment, we use 10, 20, 30, 40, 50 as degradation parameters.

The generated profile is shown in Fig. 9a with x-axis the required bandwidth and the y-axis the accuracy (F1 score). Note the log scale on the horizontal axis as raw uncompressed 8-bit RGB video streams are prohibitively large: $1920 \times 1080 \times 30 \times 3 \times 8 = 1.5$ Gbps.

Each point in the scatter plot represents one configuration that our offline profiling has evaluated. Notice the vast spread in bandwidth requirement among configurations with similar accuracy as well as the wide spread in accuracy among configurations that consumes similar bandwidth.

We first show three lines in the case of only tuning one



(a) Pedestrian Detection (b) Augmented Reality (c) Top-k

Figure 9: Application profiles.

knob. Notice the distinct behavior of the three lines. In this particular application, reducing the resolution has the most penalty because the HOG detector has a minimal 128 pixels by 64 pixels window. The camera is deployed in a far-field context; scaling down the image will quickly has an effect on the detection. Tuning frame rate doesn't affect the accuracy too much. In fact, even with 1 FPS, the accuracy is still relatively high. However, reducing frame rate doesn't bring much bandwidth saving (as we have mentioned in Fig. 4). The most effective way that reduces the bandwidth while preserving the accuracy is to adjust the quantizer because it affects almost every pixel and creates smaller P-frames.

The Pareto boundary, or *profile*, is the most important curve. In the begining it's close to the curve when only quantizer is tuned, quantizer has a certain limit. A crisp image is prefered as many image processing algorithms are looking for the edges while for human consumption, a smoother image is fine.

As the uncompressed video is not practical, we imposes a bandwidth cap before the profile is used in runtime (only use optimal configurations that creates video with less than 20mbps data rate).

Augmented Reality: We collected training set for this application ourselves. It's a 23-second video clip with 1920x1080 resolution and 30 FPS taken on a mobile phone. During the capture, we change the camera view in a slow pace to emulate how a real user would look around. Because target objects are relatively close while the camera is moving, we hypothesis for this training set, the profile will be different from the previous application that reducing frame rate will have a detrimental effect.

The generate profile is shown in Fig. 9b. First, we see our intuition backed up by measurements. Besides, the Pareto boundary also first follows the video encoding knob, but optimal settings are achieved only when multiple degradations are in effect.

Top-K: To evalute the top-k application, we generate synthetic dataset based on real-world access logs (EDGAR log file dataset, the access log of `https://sec.gov`). The original log contains CSV-format data extract from Apache web server that records and stores user access statistics [32]. The original log has only 500k access per hour; it's rather small in comparison to today's CDN log. We condensed an hour-long data into one second. After performing the local aggregation, the data size is reduced from 500k entries per second to 50k key-value pairs (10x reduction). Next we explore the space of degradation with respect to parameter N and T. The parameter N is from 100 to 15000; T from 0 to 500.

Fig. 9c shows the generated profile. As we can see, most configurations are very close to the pareto boundary. In the case when data skew is more severe, we might see that T is more severe. Regardless, with our automatic profiling tool, developers don't have to thoroughly understand the complex relationship between bandwidth, accuracy and configuration.

6.2 Runtime Performance

To evaluate the runtime behavior, we conduct controlled experiments using four geo-distributed worker nodes from Amazon EC2 (t2.large instances) and an aggregation server from our institute. For each experiment, worker nodes transmit test data for about 10 mins. During each session, we use Linux `tc` utility to adjust outgoing bandwidth to experiment with network resource variation.

We compare our system with baseline systems that directly uses TCP and UDP. In all three applications, the raw data streams are orders of magnitude larger. While our system can adapt the rate, it could be unfair to baseline solutions. We adjust the default degradation operation so that TCP and UDP would work just fine when in normal cases; in this way, we make fair comparison. In the case of UDP, shaping at the source doesn't emulate the packet loss behavior with out-of-order delivery. We use `netem` to control packet loss rate to match the desired shaping bandwidth.

In all three experiments, we see long delays in TCP. It increases linearly when the traffic shaping started. When the

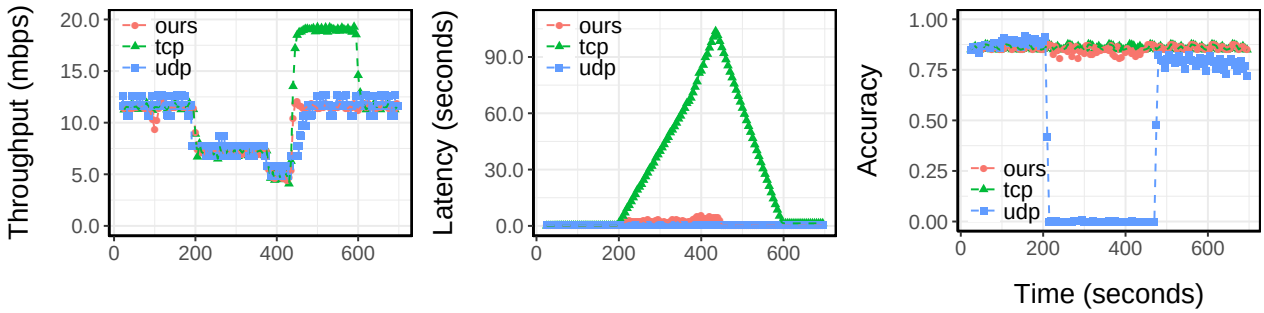


Figure 10: Runtime Adaptation of Pedestrian Detection

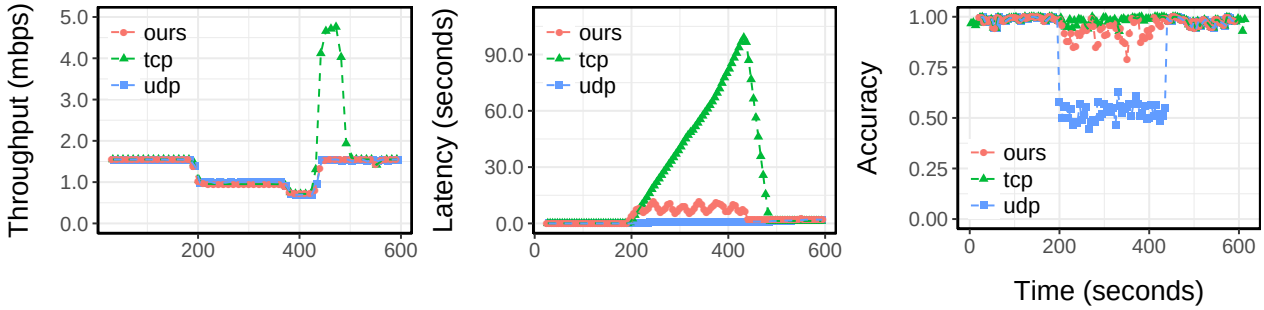


Figure 11: Runtime Adaptation of Top-K

bandwidth shaping stops, TCP quickly fills the connection to recover. Depending on the queued size, the recovery could take a few minutes or tens of seconds.

For UDP, the latency has been consistently small (mostly below 1 second) because there is no queue building up. But when traffic shaping starts, the accuracy drop is catastrophic.

Applications built with AdaptiveStream performs with a middle-ground behavior between two extremes. We notice that our delay is still on the order of ten seconds. The reason for the slow adaptation is three-folds: (1) our current implementation only requests for bandwidth information when congestion is detected, the delay of getting bandwidth estimation can be large in the case of network capacity drop; (2) we perform the bandwidth in a conservative way with smoothing to avoid sudden spikes. While more improvements are possible, the current settings are satisfactory.

7. RELATED WORK

Stream processing systems: Borealis [4], Storm [44], Streaming [48].

Approximate analytics: The idea of adapting the computation and communication is also explored in other context. BlinkDB [6] for database operations and adaptive video streaming [47]. Our work recognizes the increasing streams in the IoT context and aims to empower more general applications to benefit from adaptation. [22] we examined the problem of optimizing modern analytics services that process large quantities of geodistributed data. We presented empirical results with grouped, windowed aggregation on PlanetLab using Akamai log data, and highlighted the complexity of tradeoffs that we show are driven by several factors such as query, data, and resource characteristics.

WAN-aware: Clarinet [46], GDA [36] on geo-distributed data analytics, but not on streaming. JetStream [37] studies streaming analytics in wide area network. Using structured storage for data aggregation and explicit degradation, it demonstrates how to achieve responsiveness in the presence of bandwidth fluctuation. However, JetStream didn't explore how to automatically synthesize the degradation strategy for each application.

Adaptive Streaming: Argue that these are only for video viewing (not even video processing) and the difference. Also HLS, DASH works best with video-on-demand. For live video, RTMP is typically used; but suffer the accuracy drop.

8. DISCUSSION

We discuss other usage of AdaptiveStream and the limitations of our current system.

Wireless: In wireless communication, existing protocols doesn't work well for crowded context. For example we've all experienced slow network for applications during conference when the talk is boring. Our technique extends well into these context.

Online Profiling: Currently our profiling runs only offline. The qualify of the adaptation depends on how representative the training data is. Performing online profiling is possible. When the network capacity is enough, the aggregation server

will receive data that's not degraded. The server can run profiling as a batch job and send back the newly learned profile.

Failure Recovery: The current system doesn't explicit address the failure cases as we consider it a relevant but orthogonal problem to solve. The source node can be made failure-safe by employing multiple instances of machines. The aggregation may happen within a data center, where failure recovery can be achieve as in other stream processing systems, e.g. snapshot.

Computation: Some degradation operations will also reduce the resource demand of computation. For example image resolution drop and frame rate. However, encoding quality does not because typically processing software will decode the data first and then run the code. Exploring the trade-off between compute resource and accuracy trade-off is a relevant but orthogonal problem.

9. CONCLUSION

Whew, finally...

10. REFERENCES

[1] iPerf: The TCP/UDP bandwidth measurement tool. Accessed: 2017-01-17.

[2] One nation under CCTV: the future of automated surveillance. `http://www.wired.co.uk/article/one-nation-under-cctv`. Accessed: 2017-01-27.

[3] Skynet achieved: Beijing is 100% covered by surveillance cameras, and nobody noticed. `https://www.techinasia.com/skynet-achieved-beijing-100-covered-surveillance-cameras-noticed`. Accessed: 2017-01-27.

[4] ABADI, D. J., AHMAD, Y., BALAZINSKA, M., CETINTEMEL, U., CHERNIACK, M., HWANG, J.-H., LINDNER, W., MASKEY, A., RASIN, A., RYVKINA, E., ET AL. The Design of the Borealis Stream Processing Engine. In *CIDR* (2005), vol. 5, pp. 277–289.

[5] ABDI, H. The Kendall Rank Correlation Coefficient. *Encyclopedia of Measurement and Statistics. Sage, Thousand Oaks, CA* (2007), 508–510.

[6] AGARWAL, S., MOZAFARI, B., PANDA, A., MILNER, H., MADDEN, S., AND STOICA, I. BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. In *Proceedings of the 8th ACM European Conference on Computer Systems* (2013), ACM, pp. 29–42.

[7] ALLEN, J. G., XU, R. Y., AND JIN, J. S. Object Tracking Using Camshift Algorithm and Multiple Quantized Feature Spaces. In *Proceedings of the Pan-Sydney area workshop on Visual information processing* (2004), Australian Computer Society, Inc., pp. 3–7.

[8] ANDERSEN, M. P., AND CULLER, D. E. BTrDB: Optimizing Storage System Design for Timeseries Processing. In *14th USENIX Conference on File and Storage Technologies (FAST 16)* (2016), pp. 39–52.

[9] ATZORI, L., IERA, A., AND MORABITO, G. The Internet of Things: A Survey. *Computer networks 54*, 15 (2010), 2787–2805.

[10] BABCOCK, B., AND OLSTON, C. Distributed Top-K Monitoring. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data* (2003), ACM, pp. 28–39.

[11] BELLARD, F., NIEDERMAYER, M., ET AL. Ffmpeg. *Availabel from: http://ffmpeg. org* (2012).

[12] BRADSKI, G. The opencv library. *Doctor Dobbs Journal* (2000).

[13] CALDER, M., FAN, X., HU, Z., KATZ-BASSETT, E., HEIDEMANN, J., AND GOVINDAN, R. Mapping the Expansion of Google's Serving Infrastructure. In *Proceedings of the 2013 conference on Internet measurement conference* (2013), ACM, pp. 313–326.

[14] CANNY, J. A Computational Approach to Edge Detection. *IEEE Transactions on pattern analysis and machine intelligence*, 6 (1986), 679–698.

[15] CHUN, B.-G., IHM, S., MANIATIS, P., NAIK, M., AND PATTI, A. Clonecloud: Elastic Execution Between Mobile Device and Cloud. In *Proceedings of the sixth conference on Computer systems* (2011), ACM, pp. 301–314.

[16] COIFMAN, B., BEYMER, D., MCLAUCHLAN, P., AND MALIK, J. A Real-time Computer Vision System for Vehicle Tracking and Traffic Surveillance. *Transportation Research Part C: Emerging Technologies 6*, 4 (1998), 271–288.

[17] CORMODE, G., GAROFALAKIS, M., HAAS, P. J., AND JERMAINE, C. Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches. *Foundations and Trends in Databases 4*, 1–3 (2012), 1–294.

[18] DALAL, N., AND TRIGGS, B. Histograms of Oriented Gradients for Human Detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* (2005), vol. 1, IEEE, pp. 886–893.

[19] DOLLAR, P., WOJEK, C., SCHIELE, B., AND PERONA, P. Pedestrian Detection: An Evaluation of the State of the Art. *IEEE transactions on pattern analysis and machine intelligence 34*, 4 (2012), 743–761.

[20] EVERINGHAM, M., VAN GOOL, L., WILLIAMS, C. K., WINN, J., AND ZISSERMAN, A. The pascal visual object classes (VOC) challenge. *International journal of computer vision 88*, 2 (2010), 303–338.

[21] GRANGE, A., DE RIVAZ, P., AND HUNT, J. VP9 Bitstream & Decoding Process Specification. *Google, March* (2016).

[22] HEINTZ, B., CHANDRA, A., AND SITARAMAN, R. K. Towards Optimizing Wide-Area Streaming Analytics. In *Cloud Engineering (IC2E), 2015 IEEE International Conference on* (2015), IEEE, pp. 452–457.

[23] INDEX, C. V. N. The zettabyte era: Trends and analysis. *Cisco white paper* (2013).

[24] JOHAS TEENER, M. D., FREDETTE, A. N., BOIGER, C., KLEIN, P., GUNTHER, C., OLSEN, D., AND STANTON, K. Heterogeneous Networks for Audio and Video: Using IEEE 802.1 Audio Video Bridging. *Proceedings of the IEEE 101*, 11 (2013), 2339–2354.

[25] KRIOUKOV, A., FIERRO, G., KITAEV, N., AND CULLER, D. Building application stack (BAS). In *Proceedings of the Fourth ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings* (2012), ACM, pp. 72–79.

[26] KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. Imagenet Classification with Deep Convolutional Neural Networks. In *Advances in neural information processing systems* (2012), pp. 1097–1105.

[27] LOWE, D. G. Distinctive image features from scale-invariant keypoints. *International journal of computer vision 60*, 2 (2004), 91–110.

[28] LU, C., AND TANG, X. Surpassing Human-level Face Verification Performance on LFW with Gaussian Face. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence* (2015), AAAI'15, AAAI Press, pp. 3811–3819.

[29] MAAS, M., ASANOVIĆ, K., HARRIS, T., AND KUBIATOWICZ, J. Taurus: A Holistic Language Runtime System for Coordinating Distributed Managed-Language Applications. *ACM SIGOPS Operating Systems Review 50*, 2 (2016), 457–471.

[30] MICHALOS, M., KESSANIDIS, S., AND NALMPANTIS, S. Dynamic adaptive streaming over HTTP. *Journal of Engineering Science and Technology Review 5*, 2 (2012), 30–34.

[31] MILAN, A., LEAL-TAIXÉ, L., REID, I., ROTH, S., AND SCHINDLER, K. MOT16: A Benchmark for Multi-Object Tracking. *arXiv preprint arXiv:1603.00831* (2016).

[32] OF ECONOMIC, T. D., AND (DERA), R. A. EDGAR Log File Data Set. `https://www.sec.gov/data/edgar-log-file-data-set`. Accessed: 2017-01-25.

[33] OH, S., HOOGS, A., PERERA, A., CUNTOOR, N., CHEN, C.-C., LEE, J. T., MUKHERJEE, S., AGGARWAL, J., LEE, H., DAVIS, L., ET AL. A Large-scale Benchmark Dataset for Event Recognition in Surveillance Video. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on* (2011), IEEE, pp. 3153–3160.

[34] PANTOS, R., AND MAY, W. HTTP Live Streaming.

[35] PARKHI, O. M., VEDALDI, A., AND ZISSERMAN, A. Deep Face Recognition. In *BMVC* (2015), vol. 1, p. 6.

[36] PU, Q., ANANTHANARAYANAN, G., BODIK, P., KANDULA, S., AKELLA, A., BAHL, P., AND STOICA, I. Low Latency Geo-Distributed Data Analytics. *ACM SIGCOMM Computer Communication Review 45*, 4 (2015), 421–434.

[37] RABKIN, A., ARYE, M., SEN, S., PAI, V. S., AND FREEDMAN, M. J. Aggregation and Degradation in JetStream: Streaming Analytics in the Wide Area. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)* (2014), pp. 275–288.

[38] REDMON, J. Darknet: Open source neural networks in c. `http://pjreddie.com/darknet/`, 2013–2016.

[39] RICHARDSON, I. E. *The H.264 Advanced Video Compression Standard*. John Wiley & Sons, 2011.

[40] RIJSBERGEN, C. J. V. *Information Retrieval*, 2nd ed. Butterworth-Heinemann, Newton, MA, USA, 1979.

[41] SATYANARAYANAN, M., BAHL, P., CACERES, R., AND DAVIES, N. The Case for VM-based Cloudlets in Mobile Computing. *IEEE pervasive Computing 8*, 4 (2009).

[42] SCHULZRINNE, H. Real time streaming protocol (RTSP).

[43] TEAM, G. GStreamer: Open Source Multimedia Framework.

[44] TOSHNIWAL, A., TANEJA, S., SHUKLA, A., RAMASAMY, K., PATEL, J. M., KULKARNI, S., JACKSON, J., GADE, K., FU, M., DONHAM, J., ET AL. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data* (2014), ACM, pp. 147–156.

[45] VIOLA, P., AND JONES, M. Rapid Object Detection Using a Boosted Cascade of Simple Features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on* (2001), vol. 1, IEEE, pp. I–I.

[46] VISWANATHAN, R., ANANTHANARAYANAN, G., AND AKELLA, A. Clarinet: WAN-Aware Optimization for Analytics Queries. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (2016), USENIX Association, pp. 435–450.

[47] YIN, X., JINDAL, A., SEKAR, V., AND SINOPOLI, B. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. *ACM SIGCOMM Computer Communication Review 45*, 4 (2015), 325–338.

[48] ZAHARIA, M., DAS, T., LI, H., SHENKER, S., AND STOICA, I. Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters. In *Presented as part of the* (2012).

[49] ZHANG, B., MOR, N., KOLB, J., CHAN, D. S., LUTZ, K., ALLMAN, E., WAWRZYNEK, J., LEE, E. A., AND KUBIATOWICZ, J. The Cloud is Not Enough: Saving IoT from the Cloud. In *HotCloud* (2015).

[50] ZHANG, H., CHEN, K., BAI, W., HAN, D., TIAN, C., WANG, H., GUAN, H., AND ZHANG, M.
Guaranteeing Deadlines for Inter-Data Center Transfers.
IEEE/ACM Transactions on Networking (2016).