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. 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 [20]. 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 [2], Storm [34], or Spark Streaming [38], often focus 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 the context of wide-area, the available bandwidth easily becomes the bottleneck: it's scarce and variable [29] and not keep up with the increase rate of traffic [17].

When facing situations where the bandwidth is not sufficient, applications deployed today either choose a conservative setting to ensure a long-period running or leave the fate to the underlying transport layer. In the case of TCP, the sender will be blocked and data are backlogged, leading to severe delay. In the case of UDP, uncontrolled packet loss occurs, leading to application performance drop. While specialized adaptive streaming approaches exist for multimedia applications [24, 32], their scope is limited: the optimization target is for human consumption rather than analytics.

In this paper, we present the design and implementation of AdaptiveStream, a stream processing systems for the wide-area. The biggest difference from existing systems is to enable applications making explicit trade-off between data fidelity and freshness.

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 [28, 35]. There are two types of pre-procssing operations: lossless or lossy. While lossless transformations are helpful [29], 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 (§3.3). 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 (§3.4). 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 (§3.5).

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 deteriorate 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 [26]. 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 [14], vehicle tracking [11], or facial recognition to locate people of interest [27, 22]. **TODO: Add concrete numbers to argue for the data volume.**

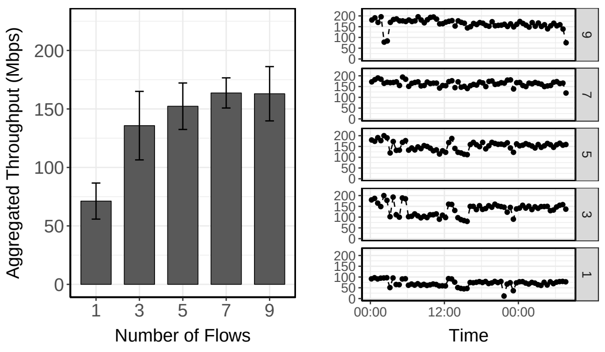


Figure 1: Bandwidth measurement between Amazon EC2 sites (from Ireland to California).

IoT Sensors: While traditional environmental sensors are slow [6], we are seeing an increasing trend with high-frequency, high-precision sensors being deployed. For example, 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 [5].

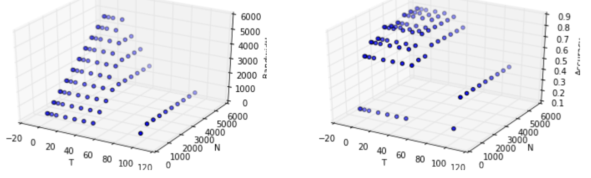
Log Analysis: Large organizations today are managing 10–100s of datacenters (DCs) and edge clusters worldwide [9]. While most log analysis today runs in a batch mode and on a daily basis, there is trend in analyzing logs in real-time for quicker optimization **TODO: cite RISE reference?**. For example, 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. While they challenge the data storage and processing system, the cloud can handle it well. The real challenge lies in the communication. Data generated from the edge, not a lot WAN bandwidth; also with cost. And worse, the bandwidth is also not guaranteed. We will demonstrate with measurement.

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 is shown in Fig. 1. Regardless of the number of flows[1], these exist occasions when the available bandwidth is almost halved. We believe the backhaul links between EC2 sites are better (if not at least representative) in comparison

[1]EC2 has a per-flow and per-VM rate limiting [40].



(a) Bandwidth requirement (b) Application Accuracy

Figure 2: How parameter affects resource requirement and application fidelity.

to the general wide-area links. The varying nature poses real challegen to the realization and successful deployment of wide-area streaming applications.

2.2 Bandwidth-Accuracy Trade-off

The edge infrastructure is capable of pre-processing the data before the communication. Data degradation. Such as frame-diff based video encoding scheme. In the case of the top-K application, we first generate windowed local counts. In our dataset, we see 100x data size reduction. While effective, these transformations are often not sufficient. In the case of top-k, there is a long tail. In the case of images/video, quantizing individual pixels will often give more space.

They help reduce the resource demand but they typically also lower the output quality. Fig. 2 shows how the image resolution affects application accuracy.

In some verticle domain, such as video encoding, adaptive scheme exists. However, there is no general solution and these solutions are not generally applicable to all applications. Most video encoding techniques will adjust the quantizer to tune the encoding size and quality; while often preserving the frame rate as these videos are for human consumption. A smooth video provides a better quality of experience than higher resolution but intermitten images.

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. local aggregation and global aggregation. [16]

This motivates us to design an application-aware rate-adapting stream processing framework for the wide area; primarily exploring the design space of degradation.

For these applications, there is a trade-off between the bandwidth and accuracy. Exploring the design space that allows explicit trading accuracy for resource constrained cases is the main goal of this paper.

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. The adaptation should maximize application utility with minimal developer input.

3.1 Challenges

There are four challenges in realizing AdaptiveStream.

C1: Diverse application and data: The best adaptation scheme is often application- and context-specific optimizations. For example, visual object detection relies on high-quality images while object tracking performs best when the temporal information between frame is preserved. While a surveillance camera in a building can work fine with lower resolution, when deployed in a far-field where target objects are already small, it's crucial not to drop the resolution.

C2: No analytical solutions: Unlike SQL queries whose demand and accuracy can typically be estimated using analytical models [12], 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 generally available.

C3: Multi-dimensional exploration: Real-world applications typically have more than one tunable parameters and 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 [19]). 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 three stages (Fig. 3).

§3.3 Developers use novel AdaptiveStream APIs to express the degradation operations and their knobs.

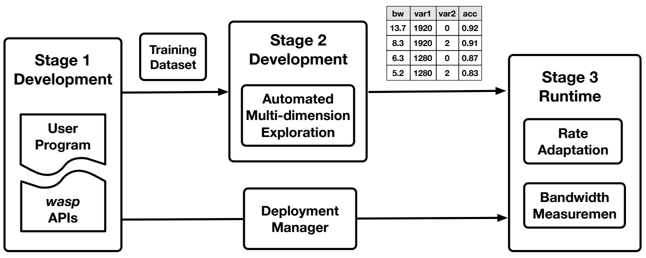
§3.4 The system performs an automatic multi-dimension profiling with representative dataset and a user-defined utility measure.

§3.5 The runtime system uses the *profile* adjusts the application execution based on live bandwidth estimation.

Before we dive into the details of our solution, we clarify that improving specific algorithms to better handle degraded data is beyond the scope of this paper.

3.3 Programming Abstraction

We first consider a strawman solution: manual policies for degradation. JetStream [29] 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 three issues with this approach.



bw	var1	var2	acc
13.7	1920	0	0.92
8.3	1920	2	0.91
6.3	1280	0	0.87
5.2	1280	2	0.83

Figure 3: System Overview

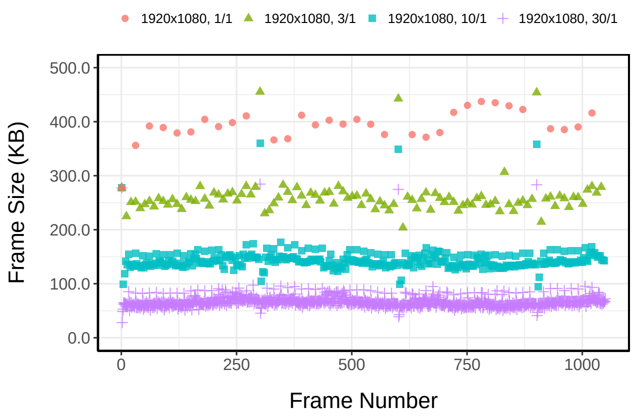


Figure 4: H.264 requires more information per frame when the frame rate is reduced.

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. Naively one would think that reducing the frame rate by 50% will half the data rate. However when video encoding is employed, the inter-frame difference is increased (P-frame size) when the frame rate is reduced. This leads to a larger data size for each frame. Fig. 4 shows an example of H.264 encoding with 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.

Fixed?

The above strawman solutions clearly requires quite some developer effort to become practical: tuning the knob to the right value and be elaborate on individual rules.

On the other hand, a completely developer-free solution is not practical, either. While static analysis can be used to identify places to optimize application execution [10], in our target applications, it is prone to false positives—exploring wrong or unnecessary parameters. With each introduced parameter to explore, the profiling time will increase drastically as this poses a combinatorial space.

The ideal programming abstraction should allow explicit annotations of degradation operations, without requiring developers being exact on the values. Think of these APIs as hints from developers: this function, together with these parameters, will likely reduce the data size and have an effect on the data fidelity; however their exact effect is not clear.

TODO: Use abstract symbols (or illustrative figures rather than Rust code here.

To realize this abstraction, we've come up with the `maybe` operator. It takes two argument: `knob` and `func` and creates a custom node in the processing pipeline called `Maybe`. We encode the constrains to the parameter using Rust's trait system. Here, `knob` can be any type T that implements the `IntoKnob` trait, which enables conversion from the type T to a `Knob` type. The `func` here takes two parameter, one of them is the item from the type `K` and the other arguments is the type of the stream item.

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

      // omitted

}
```

Developers can directly use the above API with user-defined functions, such as the following example.

```
let quantized_stream =
    stream.maybe(vec![2, 3], |knob, p| p % knob);
```

Alternatively, the API can be extended to support common operators. For example, `maybe_downsample` operator can that uses the `downsample` function internally. This function takes two argument: target size specified using a tuple and the image.

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

    // omitted

}
```

With these APIs, the example we mentioned earlier can be implemented as following.

```
let (client, server) = adastream::network();
client.stream(Camera::new(0))
      .maybe_downsample(vec![(1600, 900), (1280, 720)])
      .maybe_skip(vec![2, 5])
      .send_to(server)
      .map(|frame| frame.show())
      .compose()
```

3.4 Multi-dimensional Profiling

The second stage of AdaptiveStream involves profiling the application over all possible configurations. The goal is to select a subset of all possible degradation configurations that simutaneously minimize the data rate and maximize the accuracy (the Pareto boundary). The output *profile* is a table where each row is *(bandwidth, configuration, accuracy)*. **TODO: figure to illustrate.**

Generate configuration space: the details on how our profiling works. The profiling stage is automatic without developer intervention. When using `maybe` API, it generates an internal data structure that holds all possible values. A

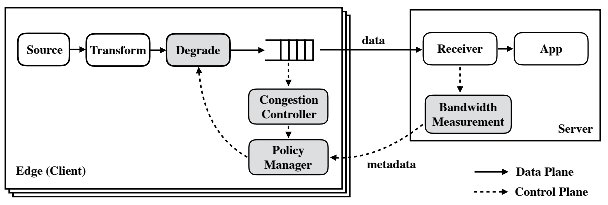


Figure 5: Runtime adaptation system architecture. The grey components are what AdaptiveStream provides.

transformation of the users program into internal data structure. Each application consists of multiple *knobs*, forming a *Configuration*.

Bandwidth measure: We measure the data that needs to be transmitted over the wire. We do not consider protocol overhead such as TCP header here.

Accuracy measure: To quantify the accuracy, we expect developers to provide a representative training dataset and an application-specific utility function that quantifies the degradation impact. The utility functions could compare the output stream with respect to some groundtruth information; or when labelled data is not available, it can compare against the original stream (with degradation not in effect). In vision applications, a typical utility function is F1 score [30]. We discuss example utilities in §4.2.

Quest for an efficient profiling method: Although exploring the entire design is embarrassingly parallel, it can be very time-consuming and resource-intensive. The impact is monotonic. We observe the following heuristics that can speed up the profiling. The degree of information loss is different along each dimension. We may optimize the space exploration by with different stride along each dimension. In general, it's not possible to save much work. **TODO: Discuss issues with the search of the entire space.** Developers can configure an accuracy bound where configurations whose accuracy is below the threshold is not considered. In this case, the applications will tolerate delays.

3.5 Runtime Adaptation

At runtime, applications automatically adapt the degradation level based on the feedback from receiver (Fig. 5.2).

Bandwidth Measurement: The receiver measures application-level througput, similar to `iPerf` [1]. To avoid sudden change, we use exponential smoothing. To avoid unnecessary communication, the client requests the measurement only when congestion is detected.

Object-level Queue: A queue where the unit is application object (frame).

Congestion monitor: Using a queue-based detection with high watermark and low watermark. This can be fixed queue length, fixed data size, or estimated queue delay based on the rate. We use the rate based.

Degradation Manager: It loads the learned profile. When receiving signals from the congestion monitor, it adjusts the degradation level to match the measured bandwidth. We leave some headroom here to allow queued objects being sent. When a congestion cleared message is received, the manager increases the rate. If some degradation operation is still in effect, after a fixed amount of time, the manager would try to reduce the level of degradation more; until it reaches the maximal allowed configuration.

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.

4. 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]

4.1 Framework

While our proposed APIs are general enough that they can be implemented in most langauges, we chose a safe language, Rust, to implement 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 [23].

Our basic APIs are similar to existing stream processing frameworks. Applications are modelled as a graph of computations where basic operators such as `window`, `map`, `filter` are provided. For brevity, we do not dive into details of these operators. Instead, we focus the discussion on our degradation operations.

```
pub trait Stream {
    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;
}
```

Our current prototype is about 2000 lines of code, with heavy use of open-source libraries.

Currently applications built with AdaptiveStream runs as a single process and application code runs in one thread communicating with the system modules using Rust's asynchronous channel primitives.

The deployment manager is using Docker container for the deployment task.

[2]Url elided for anonymity.

4.2 Building AdaptiveStream Applications

We built three applications (Fig. 6) using AdaptiveStream: pedestrian detection surveillance, an augmented reality and distributed Top-K. We only provide a brief overview of the implementation and leaves the detailed discussion to §5 together with the results.

TODO: add a table

Pedestrian Detection: This application video streams from installed CCTV cameras and detect pedestrian inside. Variant of this application can be used for safety monitoring, anomaly detection or waiting line counting. The output of the detection is the relative location (bounding box) of people in the view.

The pedestrian detection is implemented using histogram of oriented gradients (HOG) [13] with the default linear SVM classifier from OpenCV 3.1. We use the GPU-accelerated implementation for real-time processing. On a not-so-powerful GPU (GeForce GTX 970), processing 1920x1080 image takes about 50 ms.

We implemented video encoding on top of GStreamer. To be integrated with AdaptiveStream, we created a pipeline that only does encoding using H.264 (`x264enc` plugin). The pipeline exports `appsrc` and `appsink` so that we can feed raw image data in and get encoded bytes back. The GStreamer pipeline is managed in a separate and the encoding uses four threads. To ensure real-time streaming, we configured `x264enc` with `zerolatency` present and constant quality encoding. The quantizer is exported as a parameter that can be tuned.

With AdaptiveStream, this application has three degradation operations: reducing image resolution, dropping frame rate or lower video encoding quality. implement the evaluation metric using F1 score (%), which is the harmonic mean of precision and recall [30]. A successful detection is defined when the intersection over union (IOU) is greater than 50% [15].

Augmented Reality: We target at mobile augmented reality applications which offload the heavy computation to resources elsewhere. Although local computation is gaining attraction [31, 39], local wireless communication link is also susceptible to capacity variation.

For object recognition, we use a a pre-trained neural network that's trained with Imagenet [21]. Similar to our first application, we use GPU-accelerated implementation for real-time processing.

This application has the same tunable knobs with the pedestrian detection application. The application evaluation metric is also F1 score, except to be considered as a successful detection, not only the location, but also the type of the object needs to be correct.

Distributed Top-K: Many distributed system monitoring applications require to answer the "top-k" question [7], such as the top-k most popular URLs or the top-k most access files. Naive methods of sending all the raw data to the aggregation point is not feasible across the WAN. One approach to reduce the data volume is to perform local aggregation and only send the summary, such as key-value pairs of `<item, count>`. Since many real-world access patterns follow a long-tailed distribution. There is often a large-but-irrelevant tail that may be unnecessary to send. Two approaches are possible to remove the tail: (1) a local threshold (T) on the count; (2) a fixed value N that does local Top-N first. These two operators will degrade the data quality because stripped items might contribute to the aggregated top-k list.

We encode these two degradation operators with AdaptiveStream and delegate the task of finding appropriate values and runtime adaptation to it. To evaluate the degradation impact, we uses Kendall's W, which is a distance measure of the concordance between two ranked list. The output is a statistic measure from 0 (no agreement) to 1 (complete aggrement) [3].

5. EVALUATION

Our evaluation shows that the multi-dimensional profiling could explore the design space and generate profiles. With this profile and the runtime rate-based adaptation, applications built with AdaptiveStream are able to maintain a low-latency in the case of severe network degradation but also a relatively high accuracy. Under a controlled experiment, even with only transient network capacity drop, we maintain an end-to-end delay for 10 seconds in the wide-area and reasonable accuracy drop. Application-agnostic protocols creates significant backlogged data (TCP for about 100 seconds) or unusable accuracy (UDP).

5.1 Degradation Performance

We describe the training set we used in our evaluation.

Pedestrian Detection: We use MOT16 [25] to evaluate this application as our offline profiling dataset. Specifically we use MOT16-04 as the training dataset and the coresponding MOT16-03 for our runtime test data. It captures a pedestrian street at night with an elevated viewpoint. The training set has about 45.3 people per image while the test set has 69.7 on the street. The original resolution is 1920x1080, fps is 30, in total 1050 frames (35-seconds). The test set has 1500 images.

Since it would be ideal to keep the aspect ratio, our offline profiling explores resolution settings 1920x1080, 1600x900, 1280x720, 960x540, 640x320; and fps 30, 10, 5, 3, 2, 1; and video encoding quality 1, 10, 20, 30, 40, 50. The generated profile is shown in Fig. 7a. Note the log scale on the horizontal axis.

Each point on the graph represents a configuration. The pareto curve is what we will use in the runtime experiment.

We also show three additional curves that is the profile if only one of the parameter is adjusted. Tuning quantizer is closely following us that's because video encoding has been a research topic for a long time. tuning resolution quickly leads to bad behaivor becuase in this particular application, cameras are far, elevated, and human's are small. When

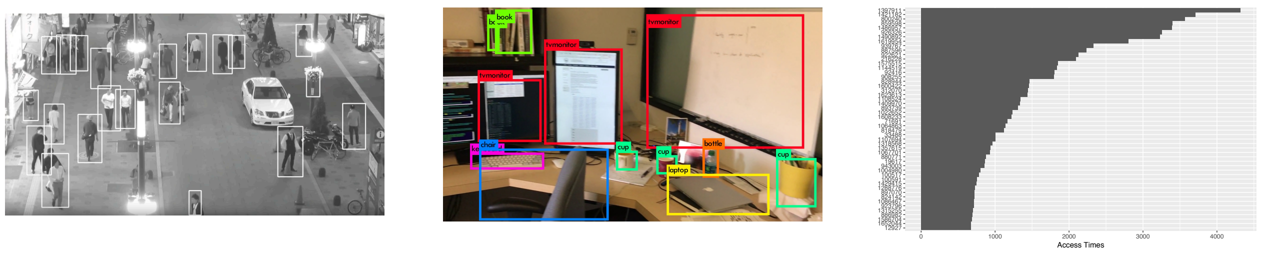
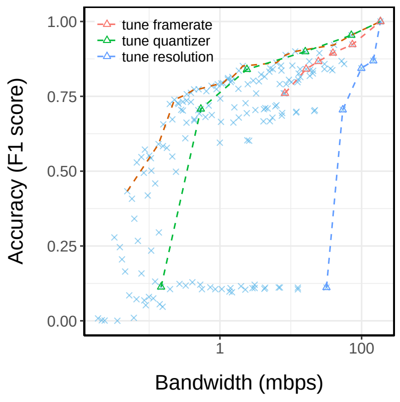
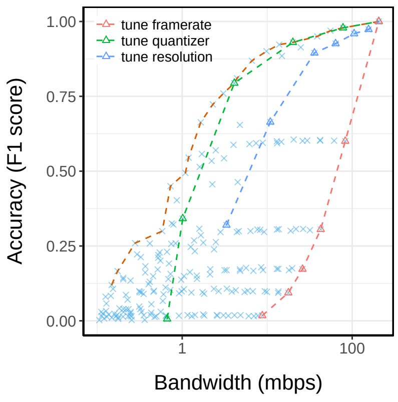


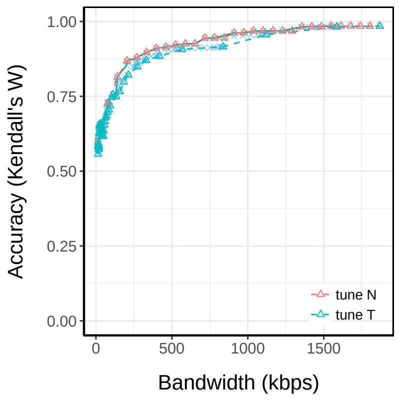
Figure 6: Three applications



(a) Pedestrian Detection



(b) Object Recognition



(c) Top-k

Figure 7: Profile. **TODO: add more legend**

reducing the resolution too much, the detection is not able to recognize them. Chanign framerate has a modest impact since pedestrians moves relatively slow.

In our runtime, we cap the bandwidth requirement to 20Mbps, removing many configurations on the right side.

Augmented Reality: The training set is a video clip of ?? seconds with 1920x1080 resolution taken on a mobile phone.

Top-K: We use Edgar log as the experiment dataset. The Division of Economic and Risk Analysis (DERA) has assembled information on internet search traffic for EDGAR filings through SEC.gov. The data is intended to provide insight into the usage of publicly accessible EDGAR company filings in a simple but extensive manner. The EDGAR Log File Data Set contains information in CSV format extracted from Apache log files that record and store user access statistics for the SEC.gov website.

This dataset is not severely skewed, so both these two configurations gives reasonable performance. Notice here, having this curve is still helpful to know exactly what N to take.

5.2 Runtime Performance

We conducted runtime experiment using geo-distributed worker nodes from Amazon EC2 and an aggregation server from our institute. Each applications runs for about 10 mins. To create a controlled environment, we use Linux `tc` utility for traffic shaping. For UDP, we emulate traffic drop instead of shaping.

For the video case, TCP is us without adaptation. UDP is using RTSP with ffmpeg.

We see a long delay in TCP. It increases linearly when the traffic shaping started. When the bandwidth shaping stops, TCP quickly fills the connection to recover. Depending on the queued size, the recovery takes a few minutes or in seconds.

For UDP, the latency has been small (mostly below 1 second). But due to packet loss, the accuracy drop is severe.

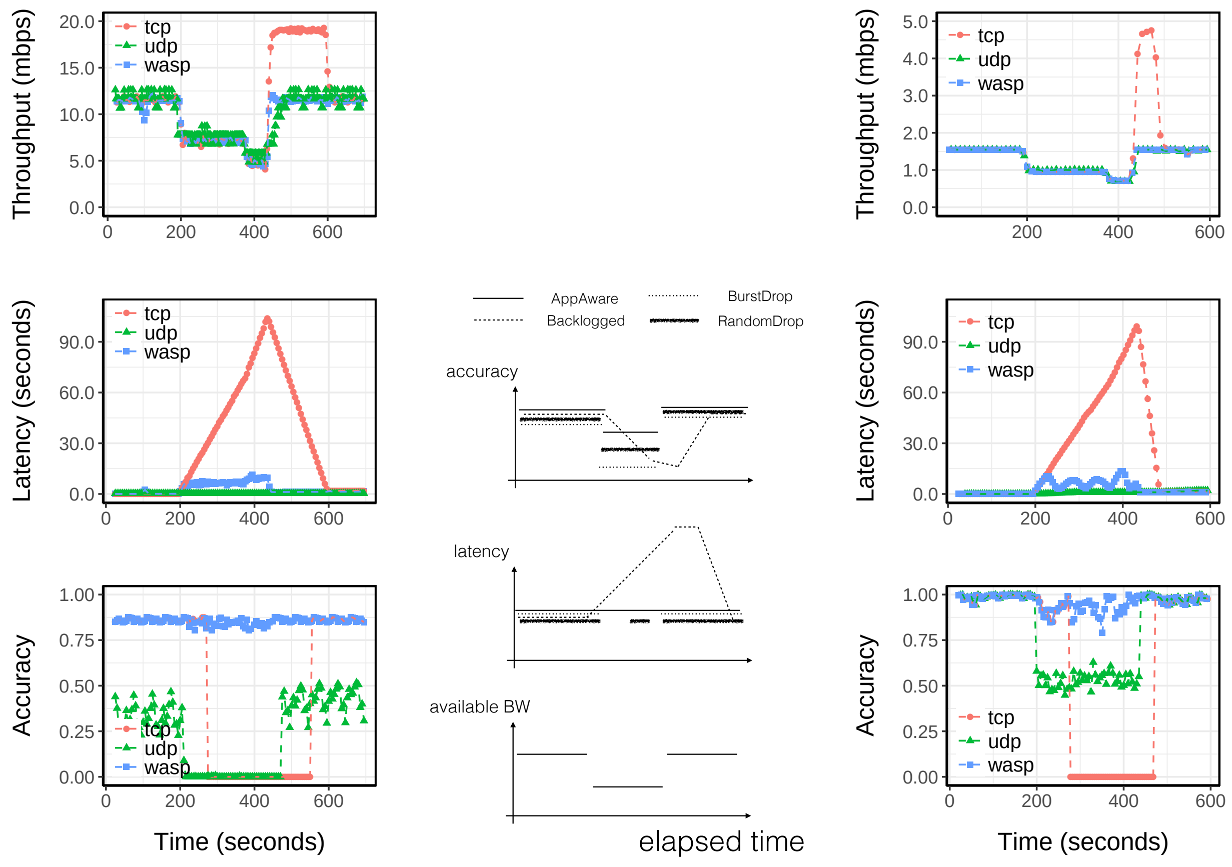
Our approach is the middle ground between these two. There is a bounded latency.

Our delay in reacting to the network change is three-folds: (1) we only request for bandwidth information when congestion is detected, the delay of getting bandwidth estimation will be large; (2) the server side estimates bandwidth in a conservative way with smoothing. While more improvements are possible, the current settings are satisfactory.

6. RELATED WORK

Stream processing systems: Borealis [2], Storm [34], Streaming [38].

Approximate analytics: The idea of adapting the computation and communication is also explored in other context. BlinkDB [4] for database operations and adaptive video streaming [37]. Our work recognizes the increasing streams in the IoT context and aims to empower more general applications to benefit from adaptation. [16] we examined the problem of optimizing modern analytics services that process large quantities of geodistributed data. We presented empiri-



(a) Pedestrian Detection (b) Object Recognition (c) Top-k

Figure 8: Runtime Adapation Behavior of AdaptiveStream. **TODO: adjust the accuracy measure**

cal 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 [35], GDA [28] on geo-distributed data analytics, but not on streaming. JetStream [29] 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.

7. 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.

8. CONCLUSION

Whew, finally...

9. REFERENCES

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

[2] 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.

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

[4] 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.

[5] 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.

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

[7] 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.

[8] BASET, S. A., AND SCHULZRINNE, H. An Analysis of the Skype Peer-to-peer Internet Telephony Protocol. *arXiv preprint cs/0412017* (2004).

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

[18] JACOBSON, V. Congestion Avoidance and Control. In *ACM SIGCOMM computer communication review* (1988), vol. 18, ACM, pp. 314–329.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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

[25] 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).

[26] 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.

[27] Parkhi, O. M., Vedaldi, A., and Zisserman, A. Deep Face Recognition. In *BMVC* (2015), vol. 1, p. 6.

[28] 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.

[29] 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.

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

[31] Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. The Case for VM-based Cloudlets in Mobile Computing. *IEEE pervasive Computing 8*, 4 (2009).

[32] Schulzrinne, H. Real time streaming protocol (RTSP).

[33] Tierney, B. TCP Tuning Guide for Distributed Applications on Wide Area Networks. *USENIX & SAGE Login 26*, 1 (2001), 33–39.

[34] 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.

[35] 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.

[36] Wright, M., Freed, A., et al. Open SoundControl: A New Protocol for Communicating with Sound Synthesizers. In *ICMC* (1997).

[37] 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.

[38] 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).

[39] 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).

[40] 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).