UC Berkeley
UC Berkeley Electronic Theses and Dissertations

Title
Adapting Swarm Applications: A Systematic and Quantitative Approach

Permalink
https://escholarship.org/uc/item/0469v8cm

Author
Zhang, Ben

Publication Date
2018

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

Adapting Swarm Applications: A Systematic and Quantitative Approach

by

Ben Zhang

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Edward A. Lee, Chair
Professor John Chuang
Professor John Wawrzynek

Summer 2018

Adapting Swarm Applications: A Systematic and Quantitative Approach

Copyright 2018

by

Ben Zhang

Abstract

Adapting Swarm Applications: A Systematic and Quantitative Approach

by

Ben Zhang

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Edward A. Lee, Chair

The swarm refers to the vast collection of networked sensors and actuators installed in our connected world. Many swarm applications generate, transport, distill, and process large streams of data across a wide area in real time. The increasing volume of data generated at the edge challenges the existing approaches of directly connecting devices to the cloud.

This thesis begins with an overview of emerging swarm and the architecture design trends for swarm applications. The swarm faces challenges from the scarce and variable WAN bandwidth and the heterogeneous compute environment (from low-power microcontrollers to powerful compute units). When network resources or compute resources are insufficient, non-adaptive applications will suffer from increased latency or degraded accuracy. Existing approaches that support adaptation either require extensive developer effort to write manual policies or are limited to application-specific solutions.

This thesis proposes a systematic and quantitative approach to build adaptive swarm applications. The solution has three stages: (1) a set of programming abstractions that allow developers to express adaptation options; (2) an automatic profiling tool that learns an accurate profile to characterize resource demands and application performance; (3) a runtime system responsive to environment changes, maximizing application performance guided by the learned profile.

We evaluate our methodology with adaptations to network resources and compute resources. For network resources, we present a complete design and implementation of the framework AWStream and several swarm applications: pedestrian detection, augmented reality, and monitoring log analysis. Our experiments show that all applications can achieve sub-second latency with nominal accuracy drops. For compute resources, we focus on improving the profiling efficiency—using Bayesian Optimization to address the large parameter space and profile transfer to address device heterogeneity.

To my family.

Contents

Contents ii

List of Figures iv

List of Tables vii

1 Introduction 1
- 1.1 Challenges with Developing Swarm Applications . . . 2
- 1.2 Systematic and Quantitative Adaptation . . . 3
- 1.3 Summary of Results and Contributions . . . 3
- 1.4 Thesis Organization . . . 4

2 Background and Motivation 6
- 2.1 The Emerging Swarm . . . 6
  - 2.1.1 Swarm Applications . . . 7
  - 2.1.2 Heterogeneous Swarm Platforms . . . 9
  - 2.1.3 Swarm and Related Concepts . . . 10
- 2.2 Massive Adoption of Cloud Platforms . . . 12
- 2.3 The Cloud is Not Enough . . . 13
- 2.4 Edge Infrastructure for the Swarm . . . 16

3 Network Resource Adaptation 18
- 3.1 Introduction . . . 18
- 3.2 Motivation . . . 21
  - 3.2.1 Wide-area Streaming Applications . . . 21
  - 3.2.2 Wide-area Bandwidth Characteristics . . . 22
  - 3.2.3 Motivation for AWStream . . . 22
- 3.3 AWStream Design . . . 24
  - 3.3.1 API for Structured Adaptation . . . 24
  - 3.3.2 Automatic Profiling . . . 27
  - 3.3.3 Runtime Adaptation . . . 28
  - 3.3.4 Resource Allocation & Fairness . . . 30

3.4 Implementation . . . 31
3.5 Evaluation . . . 33
3.5.1 Application Profiles . . . 34
3.5.2 Profiling Efficiency & Online Profiling . . . 35
3.5.3 Runtime Adaptation . . . 37
3.5.4 Resource Allocation and Fairness . . . 43
3.5.5 HTTP Live Streaming . . . 43
3.5.6 JetStream++ . . . 44
3.6 Discussion . . . 45
3.7 Chapter Summary . . . 46

4 Compute Resource Adaptation **47**
4.1 Introduction . . . 48
4.2 Motivations . . . 50
4.2.1 Heterogeneous Environment . . . 50
4.2.2 Accuracy-Time Trade-off . . . 51
4.2.3 Performance Modeling and Challenges . . . 53
4.3 Learning the Pareto-optimal Set Efficiently . . . 54
4.3.1 Programming Abstraction for Compute Adaptation . . . 54
4.3.2 Profiling Formalism . . . 55
4.3.3 Profiling Parameters with Bayesian Optimization . . . 56
4.3.4 Profile Transfer Across Devices . . . 58
4.4 Discussion . . . 59
4.5 Chapter Summary . . . 60

5 Related Work **61**
5.1 Approximation and Adaptation . . . 61
5.2 Taming Constrained Resources . . . 62
5.3 Efficient Profiling . . . 63
5.4 Related Systems . . . 64

6 Conclusion and Future Directions **65**
6.1 Thesis Summary . . . 65
6.2 Future Directions . . . 66
6.3 Reflections and Concluding Remarks . . . 67

Bibliography **68**

A A List of IoT Hacks **85**

List of Figures

2.1 Although applications usually view the cloud as the center of all connected devices (*upper diagram*), in reality the cloud is usually on the edge of the Internet backbone, just like other devices (*lower diagram*). 15
2.2 The characteristics of the mobile, the edge and the cloud. 16

3.1 The trade-off space between data freshness and fidelity when facing insufficient bandwidth (details in §3.5.3). 20
3.2 Bandwidth variations throughout the day between Amazon EC2 sites (from Ireland to California). 22
3.3 The measured bandwidth and application accuracy for two video analytics applications. (1) Manual policies lack precision without measurements and need to handle multiple dimensions, as in (a-b) and (c-d). (2) Application-specific optimizations do not generalize: degrading frame rates works well for stationary camera (a), but not for mobile camera (b). (e-h) shows example frames. 23
3.4 AWStream's phases: development, profiling, and runtime. AWStream also manages wide-area deployment. 25
3.5 Runtime adaptation system architecture. 29
3.6 Runtime adaptation algorithm. 30
3.7 Three AWStream applications: augmented reality, pedestrian detection, and distributed Top-K. 32
3.8 Intersection over Union (IOU) illustration. 32
3.9 A distributed Top-K application with two degradation operations: `head` and `threshold`. In this example, `f2`, which is not in Top-1 for either client, becomes the global Top-1 after the merge. It would have been purged if the clients use threshold T=3, demonstrating degradation that reduces data sizes affects fidelity. 33
3.10 Application profiles of three applications. Each cross point is one configuration c's performance $(B(c), A(c))$. All figures show the Pareto boundary as well as the performance if only tuning one dimension. Note the x-axis is in log scale. 34
3.11 Parallelism speeds up both offline and online profiling. The y-axis shows the profiling time for 1-second video. 36

3.12 The horizontal reference line is the target bandwidth (11 Mbps). (1) Online profiling is necessary to handle model drift ((a) vs. (b-d)). (2) Sampling techniques—partial data (c) and partial configurations (d)—can correct model drift with less profiling overhead (see Table 3.4), compared to continuous (b). We omit accuracy predictions since in all schemes AWStream finds configurations that achieve similarly high accuracy (~90%). 37
3.13 For AR, AWStream simultaneously achieves low latency and high accuracy (accuracy has a smaller variation in comparison with other baselines). 39
3.14 PD and TK performance summary. Box plot shows latency and accuracy during the traffic shaping (i.e., t=200s-440s). 41
3.15 AWStream maintains low latency and high accuracy under different network delay conditions. 42
3.16 AWStream allows different resource allocation schemes. 42
3.17 HLS setup. (Left) High-level overview: machine 1 generates a video and stores it in the server; machine 2 fetches the data and performs analytics. (Right) Detailed view: (1) FFmpeg encodes video with multiple bitrates and groups them into MPEG-TS programs; (2) `nginx-ts-module` then generates HLS chunks on the fly and stores them for nginx serving; (3) the client (using `hls.js`) periodically fetches the latest index file (`index.m3u8`) and then downloads the right chunk according to network conditions. 43

4.1 Illustration of offloading, redundant requests and compute adaptation. 49
4.2 (Left) Face detection with a photograph of the Fifth Solvay International Conference on Electrons and Photons. (Right) Processing times using Viola-Jones face detector on different machines,with default OpenCV parameters. 50
4.3 (Left) WAN network latency has variation and the latency deteriorates during downstream and upstream speed tests. (Right) Server processing latency has variation and the latency deteriorates during load increase. 51
4.4 Benchmarks for popular CNN models demonstrate the trade-off between application accuracy and processing times. For data source and details, refer to cnn-benchmarks [107]. 52
4.5 Benchmarks for Viola-Jones face detector with different parameters. These parameters form a large space that make performance modeling challenging. 53
4.6 Viola-Jones face detector parameters `VJParameters` annotated with adaptation. `VJ` implements the `Algorithm` trait. 55
4.7 Bayesian Optimization illustrated. 56
4.8 To find the Pareto-optimal set that maximizes accuracy while minimizing processing times, the acquisition criterion can suggest new samples (blue triangles) to the current Pareto-optimal set (red points) by maximizing additive epsilon (left, blue arrows) or hypervolume (right, blue shade area). This illustration is adapted from GPareto vignette [31]. 57

4.9 Using Viola-Jones as an example, BO evaluates 50 configurations and recommends 29 configurations as the Pareto-optimal boundary (the blue line in all subfigures). Greedy and Random find sub-optimal Pareto configurations with a budget of 80 evaluations (the yellow line in each figure). 58
4.10 (Left) Empirically, processing times follows a linear approximation. (Right) Stretched/-compressed profile. See paper for details. 59

List of Tables

2.1 The swarm includes a wide spectrum of computing platforms (price as of 2015). 9
2.2 Pitfalls with Today's Approach to directly using the cloud. 14

3.1 Stream processing operators in AWStream. `Vec<T>` represents a list of elements with type `T`. 25
3.2 Notations used in this chapter. 27
3.3 Application details. 31
3.4 Compared to the continuous profiling baseline (52X overhead), our sampling techniques speed up by 3× or 8.7×. 37

Acknowledgments

First and foremost, I would like to thank my advisor Prof. Edward Lee for his guidance and support in the past six years. Like most Ph.D. journeys, there were multiple occasions that I felt desperately frustrated, especially when I was stuck in a problem or could not see the value of my work. At those times, all I need was to have a meeting with Edward. He has a kind of magic that sweeps frustration away. With his breath of knowledge and ability to connect insights from diverse topics, I always left his office feeling inspired and encouraged. I have learned so much from Edward, including building proof-of-concepts (he still writes a lot of code), giving credit to people, valuing collaboration, and communicating ideas effectively. It is my great privilege to be his Ph.D. student.

During my Ph.D., I am mostly involved with the TerraSwarm project, which allowed me to collaborate with many outstanding researchers and professors. Prof. Björn Hartmann mentored me in my first project, from which I learned many practical research skills: how to brainstorm novel ideas, set a research agenda, write a paper, design graphics,[1] revise a paper, and give a conference talk. Later, with Prof. John Kubiatowicz and many other folks in the Global Data Plane project, we wrote a position paper with a fun title: "The Cloud is Not Enough." Dr. David Mellis agreed to collaborate with me after we met in a TerraSwarm poster session. Our collaboration introduced me to many software engineer practices and I became a heavy user of GitHub afterwards. Prof. John Wawrzynek offered invaluable feedback in the last few years of my Ph.D. Our discussion began with a grand SwarmOS vision and evolved into a specific direction on adaptation, i.e., this thesis.

I would like to thank Prof. Syliva Ratnasamy and Prof. John Chuang for being on my thesis committee and offering their feedback. Sylvia taught me to ask fundamental questions about the swarm and how it would change the status quo. John, with his experience in many novel applications, offered perspectives from a broader context.

I especially thank Prof. Xin Jin and feel grateful to have the chance to collaborate with him. I met him when he was a postdoc at Berkeley and was impressed by his knowledge and ideas in system research. Xin pointed me to JetStream and our discussion led to exploring systematic adaptation. He is also a master for system evaluation, a skill often considered as black art. He taught me how to design evaluation goals, run experiments, and present the results in compelling figures.

I am thankful for my undergraduate research experience that builds the foundation for my research in the swarm. I must thank my MSRA mentor Prof. Xiaofan Fred Jiang, without whom I may not have pursued a Ph.D. or come to Berkeley. In my undergraduate study, I have also received much help and guidance from Prof. Lin Zhang.

I really enjoy my stay in the DOP center. The friendly atmosphere and the inspiring conversations with colleagues/friends have helped me march on in the grinding Ph.D. experience. I am lucky to have shared 545K with Eunsuk Kang, Ilge Akkaya, Eleftherios Matsikoudis and 545N with Mehrdad Niknami. We would chat about anything when it comes to a research break. I came to Berkeley around the same time with Hokeun Kim, Marten Lohstroh, Antonio Iannopollo, and Ankush Desai. I

[1] The figure showing an augmented version of Google Glass is used on the web page of Research at Berkeley EECS.

feel sincerely grateful to have the chance to grow together with you towards independent researchers. I also feel fortunate to have met and learned from other past and current members of the Ptolemy group—Patricia Derler, Chris Shaver (Yvan Vivid), Michael Zimmer, Christos Stergiou, Dai Bui, Ben Lickly, Matt Weber, and Gil Lederman.

When it comes to logistics, Mary Stewart and Christopher Brooks are undoubtedly the best. Mary deals with the physical side and Christopher manages the cyber side. I am extremely impressed by and would like to learn Christopher's management skills, be it project, software, or life. His email "New Year's Cleanup" in 2013 was an eye-opener to me about tidying up digital assets. Thank you for keeping my graduate life easy off the boring office chores and logistics.

A great part of Edward's research group is that there are always visiting scholars. Over the past few years, I have met researchers with shockingly-broad backgrounds—Shuhei Emoto, Marjan Sirjani, Joseph Ng, Chadlia Jerad Ep Ben Haj Hmida, Moez Ben Haj Hmida, Maryam Bagheri, Victor Nouvellet, Atul Sandur, Andrés Goen, Ravi Akella, and many others.

I would also like to thank other students, researchers and collaborators I have worked with closely—Yu-Hsiang Sean Chen, Claire Tuna, Achal Dave for the Google Glass project, Ahir Reddy for a wide-area network measurement project, and Nitesh Mor, Jack Kolb, Eric Allman for the Global Data Plane.

Graduate life would have become monotonous without friends to take my mind off the stress. Thank you—Kaifei Chen, Qifan Pu, Xiang Gao, Peihan Miao, Yuting Wei, Chaoran Guo, Yiyan Ge, Tianshi Wang, Zhuo Chen, Dezhi Hong—for many memorable moments.

Berkeley and nearby cities, Emeryville, Oakland, Albany, El Cerrito, are by far the best place I have lived. Despite living on a budget as a poor graduate student, I still felt great happiness, because of many local stores: Monterey Market, Berkeley Bowl, Cheese Board, Acme Bread, Blue Bottle Coffee, Sweet Maria's, Yaoya-san, the Local Butcher Shop and more that I refrained from listing due to space constrains.

Last but not least, I am grateful to my family: my parents and my sister. While my physical distance is moving further away from home—Zhouzhi in the same town, Xi'an in the same city, Beijing in the same country, now Berkeley, on the same planet—you have always been there with your unwavering support and unconditional love. I would especially like to thank my girlfriend Limin Chen. Our love has endured the test of time, time zone difference, and long distances. Thank you for your love, company, encouragement, and sacrifices.

Chapter 1

Introduction

Over the past two decades, we have seen a growing number of networked sensors and actuators installed in our connected world. These sensors and actuators offer an unprecedented ability to monitor and act. Because of the enormous potential in solving societal-scale problems, this trend has gained significant attention, as demonstrated by many parallel efforts: Internet of Things (IoT) [23], Internet of Everything (IoE) [36], Industry 4.0 [123], The Industrial Internet [73], TSensors (Trillion Sensors) [34], Machine to Machine (M2M) [17], Smarter Planet [155], etc. In this thesis, we refer to this trend as *the swarm* because it well characterizes where the potential lies: the number and the scale of interconnected devices.

Swarm applications generate, transport, distill, and process large streams of data across the wide area, often in real time. For example, large cities such as London and Beijing have deployed millions of cameras for surveillance and traffic control [121, 197]. Buildings are increasingly equipped with a wide variety of sensors to improve energy efficiency and occupant comfort [62, 118].

With billions, potentially trillions [141], of devices interconnected, the swarm will become an enormous challenge to the underlying communication, computation, and storage infrastructure. For example, many swarm applications have adopted the cloud as a universal computation resource and storage backend [44, 92, 169, 217] because of its elasticity, economic benefits, and simplified management. However, transmitting data from sensors to the cloud requires traversing the wide-area network (WAN), which has scarce and variable bandwidth. Edge infrastructure—the fog [27, 35], cloudlet [49, 93, 172], and swarmboxes—can accompany the cloud as it reduces communication distances and provides extra resources. However, edge devices are extremely heterogeneous: some have limited processing capabilities while others can be overloaded by service requests.

The success of the swarm relies on addressing the gap between the growing application demand and resources that are scarce and variable. Facing insufficient resources, swarm applications that do not adapt will suffer. Take network resources as an example; when there is not enough bandwidth, an application may incur large latency from backlogged data if it uses TCP, or devastated data quality from uncontrollable packet loss if it uses UDP. These are extreme design points in the trade-off space between data fidelity and data freshness (as we will show in Figure 3.1). In order

to be resilient to resource availability and changes, swarm applications can adapt their behavior and change their resource demands, just as video streaming dynamically adjusts its bitrate to the available bandwidth [138].

While enabling adaptation is easy, it is challenging to come up with accurate adaptation policies that govern the application behavior in an optimal way. Developers often quickly come up with manual policies based on heuristics. While such manual policies are already a step forward compared to non-adaptive applications, they lack quantitative measurements to back up the decisions. Accurate adaptation policies often require extensive domain expertise or considerable effort. In addition, each application has its own trade-off. At a swarm scale, it is not viable to study each individual application extensively.

In this thesis, I propose to adapt swarm applications in a systematic and quantitative way. By "systematic", I mean programming abstractions that allow developers to embed adaptation options; by "quantitative", I am refering to a data-driven automatic profiling tool that learns the adaptation strategy.

Thesis Statement: *Systematic adaptation and quantitative profiling are the key to a resilient swarm.*

In the remainder of this chapter, I first summarize the challenges with developing swarm applications and elaborate our systematic and quantitative adaptation. I then highlight our key results and lay out the roadmap for the rest of the thesis.

1.1 Challenges with Developing Swarm Applications

We identify the following challenges that we need to address for a prosperity of the swarm:

- **Constrained Resources.** The swarm, with its scale, creates a large application demand that the infrastructure will fail to meet. In terms of network resources, the WAN bandwidth is unable to keep up with the pace of data generated by all different types of sensors. In terms of compute resources, end devices, especially low-power microcontrollers, cannot fulfill many computation-heavy tasks, e.g., machine learning (ML) inference.
- **Heterogeneous Devices.** The swarm has a wide spectrum of devices with heterogeneous computing capabilities, ranging from powerful computing units with specialized resources (such as GPU and TPU) to low-power microcontrollers. Applications that work on one platform will end up with a completely different performance on another platform.

A key approach to address the above challenges is to employ adaptation, i.e., adapting the application behavior to match the available and varying resource. However, to support adaptation, we need to address the following issues:

- **Large Design Space.** Swarm applications often have multiple adjustable stages or use a complex

algorithm with many tunable parameters. While these adjustable stages or tunable parameters provide flexibility in constructing swarm applications, when developing adaptation strategies, they form a large combinatorial design space.

- **Ad-hoc Development and Manual Efforts.** Existing approaches that allow adaptation require developers writing manual policies, often based on heuristics rather than quantitative analysis. These policies are not accurate; they are prone to human errors; and they lead to sub-optimal performance.

1.2 Systematic and Quantitative Adaptation

We propose to incorporate adaptation as first-class programming abstractions (systematic) and automatically learn the adaptation strategy, the profile, with a data-driven approach (quantitative). In this way, swarm applications can use the profile to guide its adaptation at runtime. This approach has three stages:

Programming Abstractions. We propose to introduce programming abstractions that are declarative instead of procedural. Developers do not need to express exactly *when* and *how* the application will adapt. Instead, they only specify *what* adaptations are available. In this thesis, we demonstrate two such programming abstractions: `maybe` for stream processing in network resource adaptation and macro annotations for algorithm parameters in compute resource adaptation.

Data-driven Automatic Profiling. Based on the available adaptation options specified by developers, we can build tools that automatically *learn* Pareto-optimal adaptation policies. To handle the large design space, we use both system techniques and statistical techniques. In this thesis, we demonstrate how parallelism and sampling speed up profiling for stream processing in network resource adaptation and how Bayesian Optimization learns near-optimal sets with substantially fewer number of samples.

Runtime Adaptation. Once the profile is learned, it is used at *runtime* to guide application adaptation, matching application demand to available resources. For example, AWStream matches the streaming data rate to the estimated available bandwidth. By avoiding congestion and using a Pareto-optimal configuration from the profile, AWStream achieves sub-second latency while maximizing the application accuracy.

1.3 Summary of Results and Contributions

In this thesis, we demonstrate a systemic and quantitative approach towards adaptation to network resources and compute resources. This thesis makes the following contributions:

Reviewing the Emerging Swarm. This thesis provides an overview of emerging swarm and the architecture design trends for swarm applications. We discuss issues with existing approaches that

directly connect devices to the cloud. We also analyze the challenges swarm applications face, with a focus on constrained resources and heterogeneous environment.

Systematic and Quantitative Adaptation. The core of this thesis is a systematic and quantitative approach for developing adaptive swarm applications. Instead of relying on manual policies, developers use well-defined programming abstractions to express adaptation options. We then use a data-driven profiling to automatically learn the profile—a set of Pareto-optimal configurations—that characterizes resource demands and application performance.

Design, Implementation, and Evaluation of AWStream. Swarm applications that communicate across the wide area need to handle the scarce and variable WAN bandwidth. We present a complete design and implementation of the framework AWStream for network resource adaptation. We have developed several swarm applications: pedestrian detection, augmented reality, and monitoring log analysis. Compared with non-adaptive approaches, AWStream achieves a 40–100× reduction in latency relative to applications using TCP, and a 45–88% improvement in application accuracy relative to applications using UDP. Compared with manual policies using a state-of-the-art system JetStream [160], AWStream achieves a 15-20× reduction in latency and 1-5% improvement in accuracy simultaneously.

Improving Profiling Efficiency. In AWStream, we demonstrated how parallelism and sampling techniques can speed up the profiling by up to 29× and 8.7× respectively. For compute resource adaptation, we have improved the efficiency further to address two challenges: Bayesian Optimization to address the large parameter space; profile transfer to address device heterogeneity. We show that BO reduces the number of profiling points by more than 50×.

1.4 Thesis Organization

The remainder of this thesis is organized as follows:

- Chapter 2 covers the background for swarm applications. I first summarize the landscape of the emerging swarm applications and show that they have fundamental differences from previous related concepts. Many swarm applications are constructed using a cloud-centric approach. I then argue against it by discussing the pitfalls, including security, privacy, scalability, latency, etc. A new tier of computing infrastructure, the edge, arises to accompany the cloud. While it reduces network latency and provides more resources, the edge has its own challenges, such as increased heterogeneity. The swarm landscape and the argument against the cloud are based on joint work with Nitesh Mor, John Kolb, Douglas S. Chan, Nikhil Goyal, Ken Lutz, Eric Allman, John Wawrzynek, Edward A. Lee, and John Kubiatowicz [223].

- In Chapter 3, I present adapting swarm applications to network resources. Many swarm applications that transport large streams of data across a wide area faces challenges from the scarce and variable bandwidth. This chapter focuses on AWStream that integrates application adaptation as a first-class programming abstraction and automatically learns an accurate profile that models

accuracy and bandwidth trade-off. Using the profile to guide application adaptation at runtime, we demonstrate that AWStream achieves sub-second latency with nominal accuracy drop (2-6%). The chapter is based on joint work with Xin Jin, Sylvia Ratnasamy, John Wawrzynek, and Edward A. Lee [221].

- In Chapter 4, I present adapting swarm applications to compute resources. Due to the heterogeneous capabilities of end devices and variable network/serving delays, it is challenging to provide consistent bounded response times for swarm applications. I propose to build a performance model that characterizes accuracy and processing time trade-off to guide the execution. This chapter focuses on efficient profiling using Bayesian Optimization (BO) to address the large parameter space and profile transfer to address heterogeneous capabilities of different devices.
- Chapter 5 surveys related research and industrial efforts.
- Finally, in Chapter 6 I conclude this thesis and identify important research directions for future work.

The research presented in this thesis is supported in part by Berkeley Ubiquitous SwarmLab; by the TerraSwarm Research Center, one of six centers supported by the STARnet phase of the Focus Center Research Program (FCRP) a Semiconductor Research Corporation program sponsored by MARCO and DARPA; by iCyPhy, the Berkeley Industrial Cyberphysical Systems Research Center, supported by Denso, Ford, Siemens, and Toyota.

Chapter 2

Background and Motivation

In this chapter, we first show the trends of swarm applications and how they differ from previous topics, such as wireless sensor networks, cyber-physical systems and ubiquitous computing. By analyzing the fundamental properties of swarm applications, we summarize the core challenges with developing swarm applications, such as constrained resources and heterogeneous environment.

One key potential with swarm applications is due to the scale facilitated by the inter-connectivity, especially as the cloud becomes mature and is used as the universal computation and storage backend. Many applications are constructed by directly connecting devices to the cloud. This trend, although understandable, is not the best long term approach. We discuss the pitfalls with this cloud-centric approach and argue that "the cloud is not enough."

To accompany the cloud, a new tier of computing infrastructure, the edge, arises. Due to its close proximity to end devices, the edge is effective to reduce network latency and provide more resource guarantees to end devices. However, unlike the cloud, the edge is resource constrained and it is unclear what capabilities we can expect from the edge. As a result, swarm applications relying on the edge need to take the heterogeneous landscape into account.

2.1 The Emerging Swarm

Smart devices are everywhere: light bulb, thermostat, mattress cover, e-diaper, fitness tracker, smart scale, smart cup, smart plate, smart fork, smart knife, flowerpot, toaster, garage door, baby monitor, yoga mat, sport-utility vehicle, and refrigerator.[1] Capable of computation and communication, they monitor and interact with the physical world. The growth of the number of devices is staggering: analysts predicted 26 billion devices by 2020, an almost 30-fold increase from 0.9 billion in 2009 [141].

We refer to the collection of sensors and actuators installed in our environment as the "swarm", a term coined by Jan Rabaey in a keynote talk at ASPDAC 2008 [159]. This term well characterizes

[1] Courtesy of Prof. Randy Katz.

where the potential lies: it is not in the individual components, but rather the scale and the number of interconnected devices. This is a shift from Moore's Law[2] to Metcalfe's Law.[3]

At Berkeley, the Qualcomm Ubiquitous Swarm Lab [198] and Terraswarm Research Center [199] were launched to address the huge potential and challenges for swarm applications in December 2011 and January 2013, respectively. Outside of Berkeley, this trend is also gaining attention, although the names are different: Internet of Things (IoT) [23], Internet of Everything (IoE) [36], Industry 4.0 [123], The Industrial Internet [73], Trillion Sensors (TSensors) [34], Machine to Machine (M2M) [17], Smarter Planet [155], etc.

Due to the widespread use of "IoT", we use "IoT" and "swarm" interchangeably in this manuscript. However, "swarm" is preferred as Lee puts it [128],

> *The term "IoT" includes the technical solution "Internet technology" in the problem statement "connected things."*

2.1.1 Swarm Applications

With billions of devices interconnected, the swarm makes a large number of applications possible. A previous survey by Atzori et al. [23] groups applications based on their domains: (i) transportation and logistics; (ii) health care; (iii) smart environment (home, office, plant); (iv) personal and social. We characterize the applications based on the usage pattern. In this way, it is easier to analyze application requirements and understand implications for the underlying system support. We group swarm applications into two general categories.[4]

Ambient Data Collection and Analytics

A wide range of swarm applications collect sensor data and monitor our environment. They operate at vastly different scales: cities [50, 177], buildings [62], homes [99], and individuals (also referred to as Quantified Self) [79, 192]. Because of the close interaction with our environment, these applications often have a direct impact on our everyday life and can help solve societal-scale problems. For example, in metropolitan cities of developing countries, such as Beijing, the air quality has deteriorated significantly. Air quality monitoring with fine granularity can advise people with appropriate actions such as wearing masks or staying at home [50].

Many environmental sensors are collecting data about physical properties that are intrinsically slow-changing, such as the occupancy, temperature, humidity, etc. As a result, these sensors sample at a low frequency, often millihertz, i.e., slower than 1 Hz. For example, in a campus-wide building instrumentation that aims to reduce energy usage [118], there are 2,145 sensors on KETI Motes: 463 CO_2 sensors, 466 humidity sensors, 460 light sensors, 290 PIR (passive infrared) sensors, and

[2]The number of transistors on a chip doubles about every two years at the same cost.

[3]The effect of a telecommunications network is proportional to the square of the number of connected users of the system, i.e., n^2.

[4]Note that these two categories are not exclusive to each other. Some applications may fall into both.

466 temperature sensors. Despite the large scale, they are configured to report data only every five seconds, yielding low bandwidth traffic. For the air quality monitoring mentioned above [50], the sensors only sample once a minute.

On the other hand, we are seeing high-frequency, high-precision sensors being deployed at scale as technology evolves and applications' requirements change. For example, energy disaggregation takes a whole building's aggregated energy signal and separates it into appliance-specific data. Recent works start to sample at 15 kHz or higher to analyze the harmonics to identify type of electrical circuitry in appliance [117] for a higher accuracy. This contrasts with prior approaches using low sampling rates (e.g., 1 Hz) and only relying on state changes [94].

Regardless of the sampling frequency and data rate, in this category, applications do not inspect the data immediately and often store it for future analytics and archival purposes. Data scientists may employ exploratory data analysis to visualize the collected data for patterns, or compare data across sensors and time for insights.

The aggregated historical data will keep increasing and create a new kind of big-data problem [67, 220], challenging our existing ways of data storage and analysis. For example, microsynchophasors, or uPMUs, monitor the electrical grid with 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 and requires specialized storage systems for efficient time-series analysis [15].

At last, because the collected data can be related with personal health or manufacture's operation status, many of these applications have serious privacy implications.

Real-time Applications with Low-Latency Requirements

Interactive applications that involve humans in the loop require a low latency response to engage the users. For example, wearable cognitive assistant [49] uses wearable devices, such as Google Glass, for deep cognitive assistance (e.g., offering hints for social interaction via real-time scene analysis). A general rule of thumb for systems that give the feeling of instantaneous response is to bound the latency to 100 ms [143, 149].

In autonomous systems where humans are not involved, a tight control over latency is important for proper and deterministic behavior [72], especially when it involves a group or feedback control. Schulz et al. [174] have summarized several latency critical uses, including factory automation, smart grids, intelligent transport systems, etc.

Satisfying applications' tight latency requirements is challenging. Timing behavior in software execution emerges from implementations rather than specifications [130]. The goal for timing in general software development is often to speed up some execution, rather than providing guaranteed and repeatable timing. When it comes to the network, the Internet provides "best-effort" service and has little quality of service (QoS) guarantees [179]. The situation can become worse as swarm applications become more complex and generate large amounts of data.

Device	CPU Speed	Memory	Price
Intel NUC	1.3 GHz	16 GB	~$300
Typical Phones	2 GHz	2 GB	~$300
Discarded Phones[5]	1 GHz	512 MB	~$22
BeagleBone Black	1 GHz	512 MB	$55
Raspberry Pi	900 MHz	512 MB	$35
Arduino Uno	16 MHz	2 KB	~$22
mbed NXP LPC1768	96 MHz	32 KB	$10

Table 2.1: The swarm includes a wide spectrum of computing platforms (price as of 2015).

2.1.2 Heterogeneous Swarm Platforms

There has been a dizzying array of embedded platforms, from powerful computing units to low-power microcontrollers (see Table 2.1). We discuss three categories below.

Microcontrollers

Examples include Arduino [19], mbed [20], and Spark [187]. This emerging category continues to expand as new open platforms appear on crowdfunding websites [111]. With an ecosystems featuring great support and libraries (e.g., Adafruit Online Tutorials [126]), people can easily add new sensors/actuators and develop applications. These platforms have unleashed the creativity of millions in building smart and interactive devices, spawning the *Maker Movement* [69].

Smartphones

As a key enabler for mobile computing, smartphones have become ubiquitous. They are equipped with an impressive set of sensors, including accelerometer, gyroscope, proximity sensor, camera, microphone, etc. The processors and displays on smartphones keep upgrading. In addition to use smartphones alone, many companies, such as Fitbit [79] and Automatic [24], use smartphones as gateways to connect low-power devices to the Internet via Bluetooth. Creative use of smartphones is also a hot topic in research, including reusing discarded smartphones [45], writing new operating systems [106], and developing novel applications [100].

Mini PCs

Ranging from powerful Intel Next Unit of Computing (NUC) to inexpensive Raspberry Pi and BeagleBone Black, these devices typically run various versions of Linux to simplify application deployment. Many companies, such as Ninja Blocks [151], SmartThings [182], and Wink [213], adopt these mini PCs with all kinds of networking capabilities—802.15.4, Bluetooth, WiFi, Ethernet—as their gateway devices. In contrast to using smartphones as gateways, mini PCs serve as nearby always-available infrastructure—the "immobiles" [195].

2.1.3 Swarm and Related Concepts

At first glance, the swarm may seem similar to other concepts such as wireless sensor network (WSN), cyber-physical systems (CPS), and ubiquitous computing (Ubicomp). These concepts all reflect a vision of how technology can affect our lives if they become deeply connected to the physical world: through large scale sensing (WSN), sense-compute-actuate (CPS), and pervasive computing (Ubicomp). In this section, we will discuss these concepts and argue that the swarm has its fundamental differences and new challenges to address.

Swarm and WSN

In the 1990s, *Smart Dust* [109] envisions wireless sensor nodes with a volume of one cubic millimeter that can sense, communicate and compute. Being so small, these devices can then be spread through our environment and enhance our interaction with the physical world. This vision quickly leads to deploying a group of spatially dispersed and dedicated wireless sensors to form a network, monitoring and recording the physical conditions of the environment.

Such wireless sensor networks (WSNs) offer the potential to dramatically advance scientific fields or extracting engineering insights with dense temporal and spatial measurements. Without exhaustively listing the rich literature [8, 228], we show a few notable deployments as follows,

- **ZebraNet** [225]. Zhang et al. deployed 7 nodes by placing sensor nodes into the zebra's collar at the 24,000-acre Sweetwaters Reserve at Kenya. The nodes traveled with zebras and measuresd GPS positions and sun/shade indication.
- **Redwoods** [200]. Tolle et al. deployed 80 nodes at the redwoods in Sonoma, California. The network recorded 44 days of air temperature, relative humidity, and photosynthetically active solar radiation data.
- **GGB** [113]. Kim et al. deployed 59 nodes over the span and the tower of the Golden Gate Bridge. The network collected ambient vibrations in two directions synchronously at 1 kHz rate for structural health monitoring.

We then compared the swarm with WSNs and show that while the swarm finds its origin in the WSN concept, the swarm represents a much broader vision, potentially connecting trillions of sensory and actuating devices that are heterogeneous and worldwide into a single platform.

WSN is application-specific; the swarm can re-purpose itself. Most WSNs focus on specific use cases. Because energy is usually the primary concern in deployed networks, the lifespan of a network benefits greatly from custom hardware design and careful tuning of software stack. After the deployment, sensor nodes are dedicated for the particular task. For the swarm, it is important to

[5]This data is from Challen et al. [45], where the original authors noted "Customer buyback price quoted by Sprint for a smartphone in good condition."

be reprogrammed on the fly for other purposes. This is best illustrated with the vision of "a tale of two smart cities" [131]: the swarm allows safe, efficient, and comfortable transportation and communication during the best of times, and secure, quick, and adaptable emergency response during the worst of times. The swarm is more efficient through sharing of resources; it is more resilient through dynamic recruiting; and it is more capable by enabling novel applications beyond developers' imagination. Albeit at the same time, the open architecture can lead to bigger security and privacy risks.

WSN nodes are significantly less heterogeneous than swarm devices. WSNs typically are comprised of sensor nodes and gateway nodes. Sensors nodes are often the same batch of hardware and form a star, tree, or mesh network after deployment. Gateway nodes are more powerful in processing capability, battery life, and transmission range, acting as data aggregation point and bridge to other networks. In comparison, each swarm node can have different types of sensors and actuators. Their processing and networking capabilities will depend on many factors, including tasks, prices, manufactures, and hardware availability.

WSNs are mostly stand-alone while the swarm interacts with the cloud and others closely. Once deployed, WSNs rarely interact with the external world, except shipping sensor data to some central storage location via the gateways. Therefore, when designing WSN, one has to carefully provision the resources. The swarm, on the other hand, coexists and closely interacts with the cloud and other essential components of the Internet. The cloud offers unbounded computational, communication and storage capacity. By offloading tasks to the cloud, the swarm can achieve what is beyond its own hardware capability.

Swarm and CPS

Cyber-physical systems (CPS), coined by Helen Gill in 2006, refers to the integration of computation (the cyber) and physical processes (the physical) [129]. CPS applications include industrial automation, manufacturing, transportation, medical devices, energy systems, and distributed robotics.

Because "cyber" and "physical" parts are tightly coupled in a feedback relation, it is important to study the intersection, or the joint dynamics, of the two for a complete understanding of the systems [161]. It is not sufficient to separately understand only one side. For many applications only in the cyberspace, i.e., general-purpose software, timing is a performance concern: a slower program is not an incorrect program and reducing the execution time is considered an optimization. In CPS, because the decision made by the cyber parts affects the physical processes and subsequent feedback reactions, a precise control of the timing is a correctness concern. Another challenge that is unique in the intersection of "cyber" and "physical" is the level of concurrency. Physical processes are composed of many parallel processes. Software, on the other hand, is often constructed as a sequence of steps. The challenge is to bridging the inherent sequential cyber world with the concurrent physical world.

The swarm can be viewed as a network of CPSs: leveraging Internet technology to connect an unprecedented number of devices, yielding a "swarm" of heterogeneous sensors and actuators that

can interact with the physical environment. As a result, the swarm also needs to address timing and concurrency issues related with physical processes.

While the swarm can reuse technologies and methodologies developed from decades of CPS research, its open architecture and the interoperability with third-party services have made the challenges more difficult. For example, in a standalone CPS development, one can use specialized networks with synchronized clocks and time-division multiple access (TDMA), including CAN busses, FlexRay [56], and TTEthernet [189], to provide latency and bandwidth guarantees [130]. When this CPS interacts with the public Internet and packets need to traverse the wide area network, the guarantees are no longer possible. When swarm applications cannot change the underlying infrastructure due to its openness and interoperability, they can adapt to available resources. This thesis studies how to adapt swarm applications in depth.

Swarm and Ubicomp

Ubiquitous computing (or "ubicomp") refers to the vision when computing is made to appear anytime and everywhere, so ubiquitous that no one will notice their presence [212].

Ubicomp is more than just technology. It is a paradigm shift that challenges how human interact with the environment with unnoticeable and invisible computing. As a result, ubicomp raises many human-centered challenges, such as interaction techniques for usability. For example, in a smart building that has many appliances with computing power and controls, it becomes hard to even select and control them. Instead of browsing a list, users would prefer voice commands [10] or direct manipulation—"What you see is what you can control" [47, 222].

The swarm and IoT have largely evolved out of ubiquitous computing, with a focus on the scale and interconnectivity. A standalone environment rich in computing resources for human use is still considered ubiquitous computing. In contrast, the swarm often bridges disparate components and covers the interaction between devices and environment.

2.2 Massive Adoption of Cloud Platforms

Cloud Computing [21] delivers computing services over the Internet, offering computing as a utility. The term "cloud" often refers to the hardware and systems in the datacenters that provide those computing services. A cloud can be made public such that customers use it in a pay-as-you-go manner, such as Amazon Web Services, Google AppEngine, and Microsoft Azure. In this way, the cloud offers a number of advantages over traditional IT approaches.

1. **Cost.** The cloud eliminates up-front capital expenses and resource management. Cloud users can focus on their application logic without hiring IT experts to manage on-site datacenters. It saves cloud users many time-consuming IT chores, including provisioning, physical configuration, hardware upgrades, software patching, etc.

2. **Elasticity.** The cloud offers an illusion of infinite resources on demand. It allows cloud users to start small without planning far ahead. Cloud users can increase their resources when the demand grows and decrease to reduce costs. They can also request specialized hardware resources, such as FPGA, GPU, and TPU [2].

3. **Utilization.** The pay-as-you-go model incentives users to release the resources that are no longer needed. As a result, these resources can be reused by cloud providers. Taking advantage of statistical multiplexing, the cloud can greatly improve resource utilization.

4. **Availability.** The cloud facilitates data availability and sharing. All end users can access the services "anytime, anywhere", making collaboration seamless. In addition, the cloud keeps the data safe and durable (with backups and disaster recovery).

Because of these benefits, cloud computing has shaped the software industry and made the development and deployment of web services easier than ever. It has become the *de facto* central piece around which modern web applications are constructed.

Riding on this popularity of the cloud, many of today's solutions for the swarm and IoT arise by connecting embedded platforms to the cloud as a universal computation and storage backend. Many companies have taken this approach, such as Carriots [44], GroveStreams [90], SAMI [169], Xively [214]. They offer easy-to-use APIs, data processing, visualization, and sample code for various hardware platforms. On the other hand, academic research also explores what the cloud can offer [92, 217]. For example, Bolt [92] provides data management for the Lab of Things (LoT) [39] and uses Amazon S3 or Azure for data storage.

2.3 The Cloud is Not Enough

At first glance, directly using the cloud seems to be a natural architecture for swarm applications. The industry can benefit hugely from the economic model of the cloud. With little investment in infrastructure, even novice users can start collecting sensor data and streaming it back to the cloud. However, several significant problems with this approach are revealed on closer inspection, including issues with privacy, security, scalability, latency, bandwidth and availability. While these problems are not new to typical web applications, they are exacerbated in the swarm space because of fundamental differences between swarm and web services. We summarize these concerns in Table 2.2 and list the reasoning as follows:

1. **Privacy and Security.** Sensors implanted in our surrounding environment collect extremely sensitive information. In a talk given by Wadlow [208], he described the IoT as "hundreds of computers that are aware of me, can talk about me, and are out of my control." This is a strong call for intrinsic security and privacy. This need is echoed in critical posts and talks—"Internet of Crappy Things" [70], "The Internet of Fails" [188], etc. The FTC's Technical Report [81] also emphasizes security in the IoT spaces. The list in Chapter A demonstrates the severity of

	Web/IT	Swarm/IoT
Privacy & Security	Open for access	Sensitive data
Scalability	Power law	Billion devices
Interaction Model	Human	Machine
Latency	Variable	Bounded
Bandwidth	Downstream	Upstream
Availability (QoS)	No guarantee	Requirement
Durability Management	Cloud controls	Users control

Table 2.2: Pitfalls with Today's Approach to directly using the cloud.

the security landscape. On the other hand, as a centralized resource out of users' control, the cloud presents an ever-present opportunity to violate privacy. Today, privacy has become a luxury [16], a situation that will be exacerbated in the IoT.

2. **Scalability.** By 2020, Cisco estimates 50 billion [75] devices will be connected to the cloud, while Gartner estimates 26 billion [141]. Scalability in the IoT spaces will be more challenging than web-scale or Internet-scale applications; the amount of data generated easily exceeds the reported trillion objects in Amazon S3 [28]. The bisection bandwidth requirements for a centralized cloud solution are staggering, especially since most data acquired by IoT devices can or should be processed locally and discarded.

3. **Modeling: Peripheral devices are physical.** Both sensors and actuators are physically present devices in our environment. Although sensor data can be collected and replicated (similar to virtualizing sensors [216]), the data is still generated from the edge of the network. Moreover, actuators cannot be virtualized and oftentimes the actuations cannot be rolled back [61]. This is significantly different from the model of web services today.

4. **Latency: The cloud model differs from reality.** Application developers view the cloud as a component that interconnects the smart devices. However, from a network point of view, the cloud is on the edge of the network (see Figure 2.1). Even simple IoT applications, such as those that turn on a fan in response to a rise in local temperature, will experience unpredictable latencies from sensing, wireless transmission, gateway processing, Internet Service Provider (ISP), Internet, and cloud processing.

5. **Bandwidth: Upstream traffic dominates.** Shipping data to the cloud incurs a significant amount of upstream traffic. Typical broadband networks have more downstream bandwidth than upstream bandwidth. IoT applications, however, generate data at the edges of the network, a pattern that will easily saturate the upstream link bandwidth—especially at scale. For example, a single Dropcam requires "a high speed internet connection with at least 0.5 Mbps" to use its service [85]. Even simple sensors, such as energy meters, can benefit from a higher sampling rate (the motivation of 1 kHz energy data with ground-truth from the

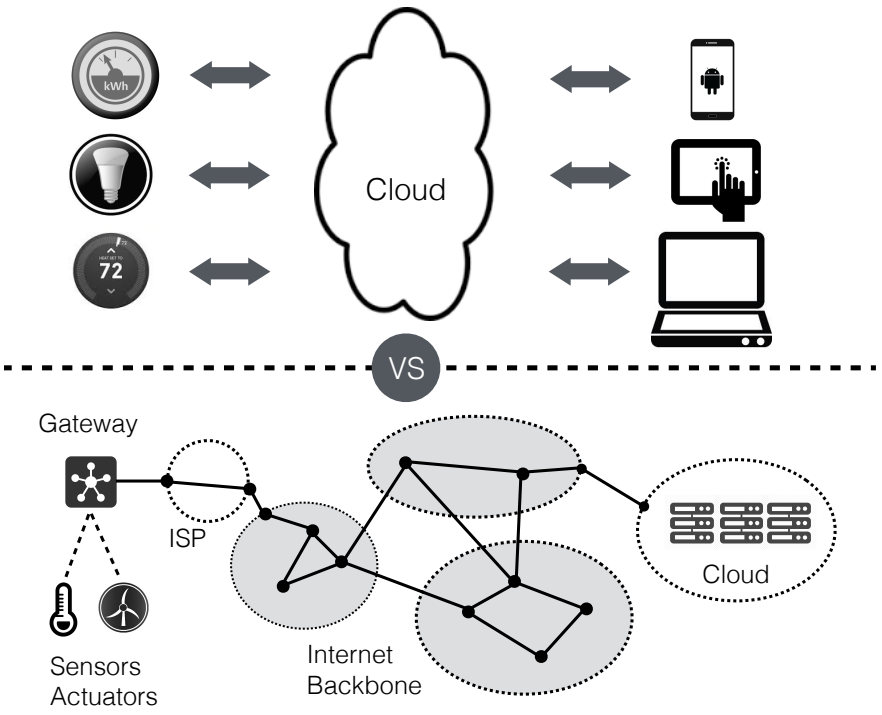


Figure 2.1: Although applications usually view the cloud as the center of all connected devices (*upper diagram*), in reality the cloud is usually on the edge of the Internet backbone, just like other devices (*lower diagram*).

Ubicomplab at the University of Washington [91] and 15 kHz sampling of energy from MIT REDD Dataset [117]).

6. **Quality of Service (QoS) Guarantees.** Web users tolerate variable latency and occasional loss of web services. In contrast, the temporary unavailability of sensors or actuators within IoT applications will directly impact the physical world. While significant engineering effort has been put into improving the availability and latency profile of the cloud (allowing Service Level Agreements), such efforts are stymied by operator error, software bugs, DDoS attacks, and normal packet-to-packet variations from wide-area routing. Further, the Internet connection to people's homes is far from perfect. Over 10% of home networks in the developed world see connectivity interruptions of more than ten minutes more frequently than once every 10 days [89]; this situation is worse in developing countries.

7. **Durability Management.** Some sensor data is ephemeral: while other data should be durable against global disasters. For ephemeral data, there is no effective way of verifying the data has been completely destroyed because the cloud is out of the user's control. For durable data, regardless of the promised guarantees [13], the reliability of cloud storage remains a major concern and there is active research in this direction [30]. Moreover, whatever durability is

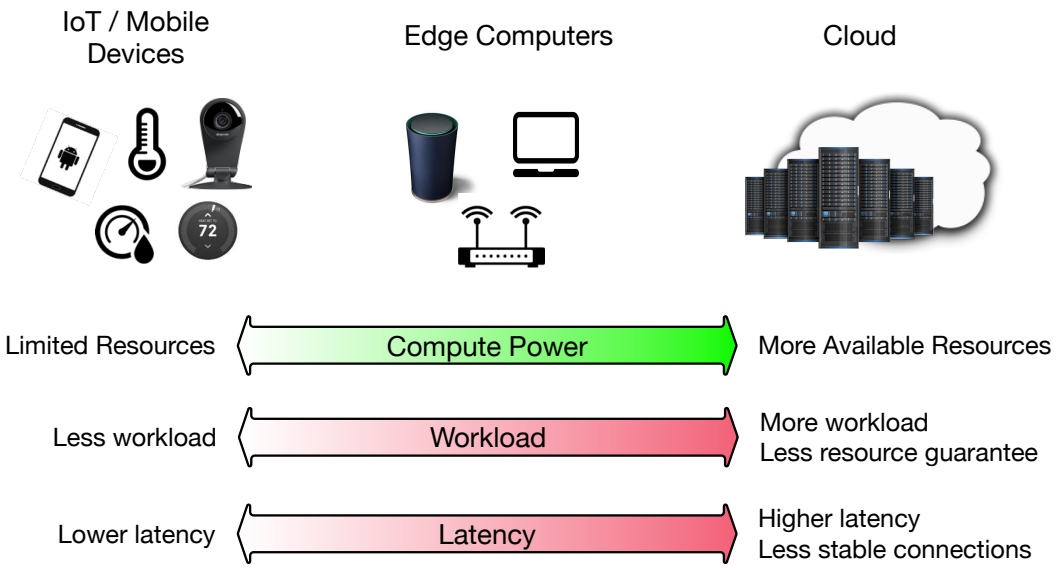


Figure 2.2: The characteristics of the mobile, the edge and the cloud.

achieved by the cloud, it is typically done so without concern for application-specific privacy or export rules. Note that control over durability is closely related to control in general: making sure that users retain control over their data rather than providers.

2.4 Edge Infrastructure for the Swarm

The edge represents a new tier of infrastructure envisioned to address some pitfalls with the cloud for IoT applications. Cisco named it *the fog* because the fog is a cloud close to the ground [35]. CMU chose *cloudlet* to indicate this small-scale cloud datacenter [93, 172]. At Berkeley, we use *swarmbox* to name the hardware platform that accompanies swarm devices. Both Smartphones and Mini PCs mentioned in §2.1.2 can act as edge computing platforms. Other edge computing infrastructure may be provided by carriers, hosted in central offices[6] or cell towers [80].

One common form of the edge computers are gateways. Many gateways provide application-specific connectivity for IoT devices to interact with the Internet, bridging low-power communication protocols such as BLE/802.15.4 to IP. Even when devices can utilize more standard communication protocols such as WiFi, the gateways still exist as downloadable applications that run on cellphones or computers, providing custom services or user interfaces.

Figure 2.2 illustrates the characteristics of the three-tier architecture. The edge is much more powerful than end devices but less so compared with the highly-centralized cloud. The edge serves a local area, such as homes, buildings, or a city. As a result, it has moderate workload. The main benefit

[6]Often, a central office is defined as a building used to house the inside plant equipment of potentially several telephone exchanges, each serving a certain geographical area.

of edge computers comes from the prime location: it sits in the middle between IoT/mobile devices and the cloud. Applications built with the edge can reduce network latency, keep data/information local, and tolerate cloud service outage.

Researcher have begun to explore the benefit of edge platforms. For example, Kim proposes an approach that leverages the edge computers as locally centralized points for authentication and authorization to address IoT security [112]. Zhuo demonstrates how the edge infrastructure can support computation offloading to achieve low latency [49]. Mor et al. proposes a data-centric design that focuses around the distribution, preservation, and protection of information to address data privacy, scalability, durability, etc [144].

While an open edge platform can realize these benefit [217], companies tend to provide their own gateways, such as Ninja Sphere [151], SmartThings Hub [182], Wink Hub [213]. The fact that custom gateways are an integral part of swarm applications leads directly to "stovepipe" solutions or balkanization. Data and services from one company cannot be shared or utilized by devices from another company: connection protocols, data formats, and security mechanisms (when present) are proprietary and often undocumented. How to address the stovepipe issue is an active research topic. For example, Brooks et al. [38] proposes a component-based software architecture named "Accessors" that are proxies for services and things.

In addition to an open architecture, we also need to address the heterogeneous capabilities of the edge infrastructure. As mentioned in Table 2.1, the swarm includes a wide spectrum of devices. It is unclear at development time about which specific device serves as the edge. It is also difficult to make changes to the edge, especially when it is a gateway. This is significantly different from the cloud. With its elasticity, the cloud offers the illusion of infinite compute resources. Users can easily start another machine or even a GPU-enabled one for heavy computation instantly. For gateways, users will need to purchase additional devices or upgrade their existing ones: it easily takes hours or days.

In summary, the edge offers a unique opportunity to compensate the cloud. However, to fully realize its potential, we need to avoid stovepipe solutions and address challenges with the heterogeneous capabilities. This thesis will focus on the latter.

Chapter 3

Network Resource Adaptation

In this chapter, we focus on swarm applications adapting to network resources. Specifically, we target at swarm applications that rely on wide-area streaming and perform analytics elsewhere, e.g., the cloud. These streaming applications face the challenge of scarce and variable wide-area network (WAN) bandwidth. Many applications today are built directly with TCP or UDP, and they suffer from increased latency or degraded accuracy. State-of-the-art approaches that adapt to network changes require developer writing sub-optimal manual policies or are limited to application-specific optimizations.

We present AWStream, a stream processing system that uses adaptation to simultaneously achieve low latency and high accuracy in the wide area, requiring minimal developer efforts. To realize this, AWStream follows our proposed methodology: (i) it integrates application adaptation as a first-class programming abstraction in the stream processing model; (ii) with a combination of offline and online profiling, it automatically learns an accurate profile that models accuracy and bandwidth trade-off; and (iii) at runtime, it carefully adjusts the application data rate to match the available bandwidth while maximizing the achievable accuracy. We evaluate AWStream with three real-world applications: augmented reality, pedestrian detection, and monitoring log analysis. Our experiments show that AWStream achieves sub-second latency with only nominal accuracy drop (2-6%).

3.1 Introduction

Among swarm applications, wide-area streaming analytics are becoming pervasive. Large cities such as London and Beijing have deployed millions of cameras for surveillance and traffic control [121, 197]. Buildings are increasingly equipped with a wide variety of sensors to improve energy efficiency and occupant comfort [118]. Geo-distributed infrastructure, such as content delivery networks (CDNs), analyze requests from machine logs across the globe [145]. These applications all transport, distill, and process streams of data across the wide area, in real time.

A key challenge that the above applications face is dealing with the scarce and variable bandwidth

in the wide area [101, 205]. As many have observed, WAN bandwidth growth has been decelerating for many years while traffic demands are growing at a staggering rate [147, 148, 196]. In addition, scarcity in last-mile bandwidth remains a problem across wireless [33], cellular [150], and even broadband [89, 191] networks. Finally, as we elaborate on in §4.2, not only is WAN bandwidth scarce, it is also relatively expensive, and highly variable.

For all of the above reasons, it is important that streaming applications be *adaptive*, incorporating the ability to optimally trade-off accuracy for bandwidth consumption and hence a key system challenge is to design the *programming abstractions and tools* that simplify the development of such adaptive applications.

In recent years, systems such as Storm [201], Spark Streaming [218], and VideoStorm [224], have emerged in support of stream processing. These systems enable efficient processing of large streams of data, but are designed to work within a single datacenter cluster (where network bandwidth is typically not the bottleneck) and hence they do not focus on support for adapting to the vagaries of WAN bandwidth.

Recent research on WAN-aware systems promote pushing computation to the network edge [160, 172]. However, even with edge computing, the need for adaptation remains because end-devices such as cameras and mobile phones still suffer from limited bandwidth in the last-hop infrastructure [3, 226]. In addition, edge computing is not a panacea as wide-area communication is often not entirely avoidable: e.g., some analytical jobs require joining or aggregating data from multiple geo-distributed sites [158, 204], while in some cases processing benefits substantially from specialized computing resources such as GPUs and TPUs [2] in the cloud.

The core difficulty with adaptive streaming analytics is that, when bandwidth is scarce, developers are faced with the decision of how to reconcile data fidelity (i.e., not losing any data) with data freshness (i.e., sending data as quickly as possible). A deterioration in either fidelity or freshness can impact application accuracy but the exact impact varies depending on the application.[1] Figure 3.1 illustrates this trade-off with a few sample points in the design space.

Applications that simply use existing protocols without any attempt at adaptation can result in extreme design points. For example, streaming over TCP ensures reliable delivery (hence high fidelity) but backlogged data delays the delivery of data (hence freshness suffers). On the other hand, streaming over UDP minimizes latency by sending packets as fast as possible, but uncontrolled packet loss can devastate data fidelity.

Manual policies, such as sampling, allow developers to trade data fidelity for freshness [160]. However, it's difficult to write accurate policies without extensive domain expertise or considerable effort. In practice, developers write manual policies based on heuristics rather than quantitative measurements and, as we show in §3.5, such policies can lead to sub-optimal performance in terms of both freshness and fidelity.

Furthermore, application-specific optimizations often do not generalize. A fine-tuned adaptation

[1] For example, an application tracking the current value of a variable might prioritize freshness while one that is computing an average might prioritize fidelity.

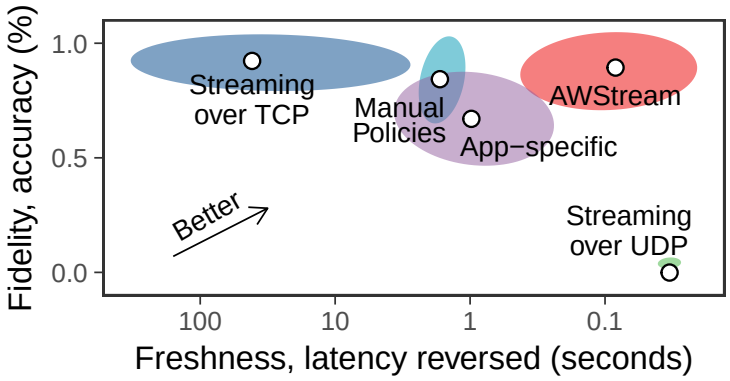


Figure 3.1: The trade-off space between data freshness and fidelity when facing insufficient bandwidth (details in §3.5.3).

algorithm for one application works poorly for a different application, if performance metrics or data distributions change. For example, video streaming focuses on quality of experience (QoE) [138, 156, 215]. Because humans favor smoothness over image quality, these systems maintain a high frame rate (e.g., 25 FPS), and reduce the resolution under bandwidth limitation. However, low resolution images can lead to poor accuracy for video analytics that rely on the image details (e.g., face detection [203]).

In this chapter, we present AWStream, a framework for building adaptive stream processing applications that simultaneously simplifies development *and* improves application accuracy in the face of limited or varying wide-area bandwidth. AWStream achieves this with three novel contributions:

1. AWStream introduces new programming abstractions by which a developer expresses *what* degradation functions can be used by the framework. Importantly, developers do not have to specify exactly when and how different degradation functions are to be used which is instead left to the AWStream framework.

2. Rather than rely on manual policies, AWStream automatically *learns* a Pareto-optimal policy or strategy for when and how to invoke different degradation functions. For this, we design a methodology that uses a combination of offline and online training to build an accurate model of the relationship between an application's accuracy and its bandwidth consumption under different combinations of degradation functions. Our solution exploits parallelism and sampling to efficiently explore the configuration space and learn an optimal strategy.

3. AWStream's final contribution is the design and implementation of a runtime system that continually measures and adapts to network conditions. AWStream matches the streaming data rate to the measured available bandwidth, and achieves high accuracy by using the learned Pareto-optimal configurations. Upon encountering network congestion, our adaptation algorithm increases the degradation level to reduce the data rate, such that no persistent queue builds up.

To recover, it progressively decreases the degradation level after probing for more available bandwidth.

We implement AWStream and use it to prototype three streaming applications: augmented reality (AR), pedestrian detection (PD), and distributed Top-K (TK). We use real-world data to profile these applications and evaluate their runtime performance on a geo-distributed public cloud. We show that AWStream's data-driven approach generates accurate profiles and that our parallelism and sampling techniques can speed up profiling by up to 29× and 8.7× respectively.

With the combination of AWStream's ability to learn better policies and its well-designed runtime, our evaluation shows that AWStream significantly outperforms non-adaptive applications: achieving a 40–100× reduction in packet delivery times relative to applications built over TCP, or an over 45–88% improvement in data fidelity (application accuracy) relative to applications built over UDP. We also compare AWStream to JetStream [160], a state-of-the-art system for building adaptive streaming analytics that is based on manual policies. Our results show that besides the benefit of generating optimal policies *automatically*, AWStream achieves a 15-20× reduction in latency and 1-5% improvement in accuracy simultaneously relative to JetStream.

3.2 Motivation

In this section, we first examine the gap between high application demands and limited WAN bandwidth. We then show that neither manual policies nor application-specific optimizations solve the problem.

3.2.1 Wide-area Streaming Applications

We focus on wide-area streaming analytics, especially the emerging IoT applications. We give two concrete examples.

Video Surveillance. We envisage a city-wide monitoring system that aggregates camera feeds, from stationary ground cameras and moving aerial vehicles, and analyzes video streams in real time for surveillance, anomaly detection, or business intelligence [152]. Recent advances in computer vision have dramatically increased the accuracy for automatic visual scene analysis, such as pedestrian detection [68], vehicle tracking [55], and facial recognition to locate people of interest [134, 157]. While some surveillance cameras use dedicated links, an increasing number of surveillance systems, such as Dropcam [85] and Vigil [226], use the public Internet and wireless links to reduce the cost of deployment and management.

Infrastructure Monitoring. Large organizations today are managing tens of datacenters and edge clusters worldwide [40]. This geo-distributed infrastructure continuously produces large volumes of data such as data access logs, server monitoring logs, and performance counters [9, 158, 205]. While most log analysis today runs in a batch mode on a daily basis, there is a trend towards analyzing logs in real time for rapid optimization [160]. For example, CDNs can improve the overall efficiency

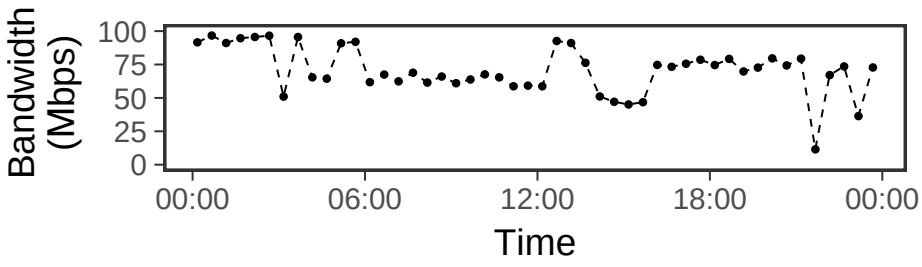


Figure 3.2: Bandwidth variations throughout the day between Amazon EC2 sites (from Ireland to California).

by optimizing data placement if the access logs can be processed in real time. In Industrial IoT, large-scale real-time sensor monitoring is becoming pervasive to detect anomalies, direct controls, and predict maintenance [26, 82].

3.2.2 Wide-area Bandwidth Characteristics

WAN bandwidth is insufficient and costly, as shown by other systems [101, 158, 205, 207]. Using Amazon EC2 as a case study, the WAN bandwidth capacity is 15x smaller than their LAN bandwidth on average, and up to 60x smaller in the worst case [101]. In terms of pricing, the average WAN bandwidth cost is up to 38x of the cost of renting two machines [11, 101].

In addition to the scarcity and cost, the large variability of WAN bandwidth also affects streaming workloads. We conducted a day-long measurement with iPerf [74] to study the pair-wise bandwidth between four Amazon EC2 sites (N. California, N. Virginia, Tokyo, Ireland). The results show large variance in almost all pairs—Figure 3.2 is one such pair. There are occasions when the available bandwidth is below 25% of the maximum bandwidth.

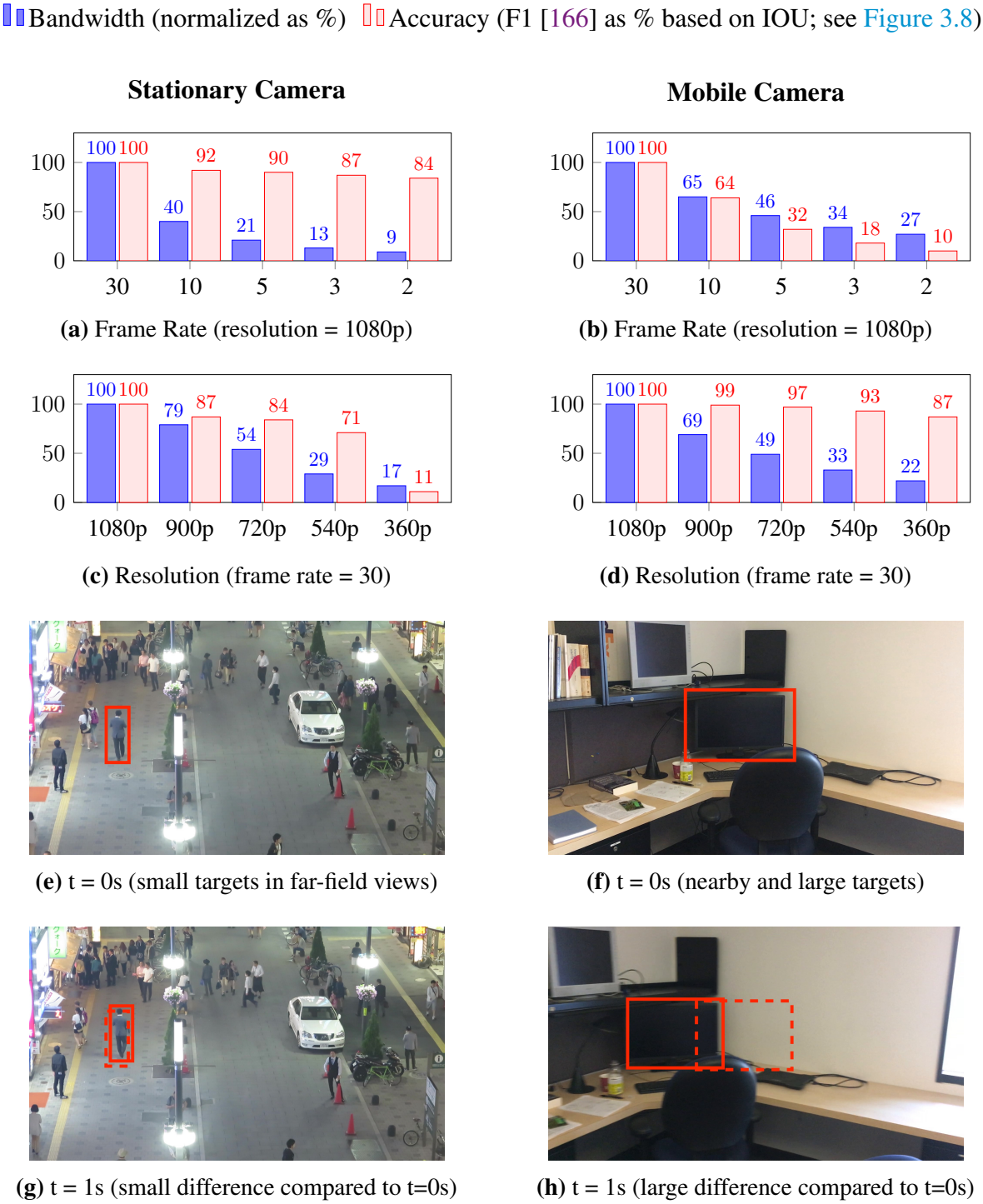
The back-haul links between EC2 sites are better—if not at least representative—in comparison to general WAN links. Similar scarcity and variations exist in wireless networks [33], broadband access networks [89, 191] and cellular networks [150].

3.2.3 Motivation for AWStream

To address bandwidth limits, existing solutions use manual policies or application-specific solutions. We discuss their drawbacks to motivate AWStream (design in §3.3).

Manual polices are sub-optimal. JetStream [160] is the first to use degradation to address bandwidth limits in wide area. While effective in comparison to non-adaptive systems, JetStream requires developers to write manual policies, for 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 image fidelity."*[2] We discuss the problems with manual policies below and present quantitative evaluations in §3.5.3.

[2]Excerpt from JetStream §4.3 [160].



(a) Frame Rate (resolution = 1080p)

(b) Frame Rate (resolution = 1080p)

(c) Resolution (frame rate = 30)

(d) Resolution (frame rate = 30)

(e) t = 0s (small targets in far-field views)

(f) t = 0s (nearby and large targets)

(g) t = 1s (small difference compared to t=0s)

(h) t = 1s (large difference compared to t=0s)

Figure 3.3: The measured bandwidth and application accuracy for two video analytics applications. (1) Manual policies lack precision without measurements and need to handle multiple dimensions, as in (a-b) and (c-d). (2) Application-specific optimizations do not generalize: degrading frame rates works well for stationary camera (a), but not for mobile camera (b). (e-h) shows example frames.

First, this policy is not accurate. Developers write such rules based on heuristics and do not back them up with measurements. Images with 75% fidelity do not necessarily lead to 75% application accuracy. In terms of bandwidth, naively one would think that reducing the frame rate by half will also half the data rate. But if video encoding such as H.264 [165] is used, a reduction in frame rate increases the inter-frame difference and creates P-frames with larger sizes. Figure 3.3b shows that when reducing the frame rate to 33% (from 30 FPS to 10 FPS), the bandwidth use can still be more than 60%.

Second, it is not scalable to specify rules one by one. A fine-grain control requires many rules in the policy. Besides, applications can degrade in multiple dimensions and each dimension has different impacts (compare Figure 3.3a with Figure 3.3c). Specifying rules in detail and across dimensions manually is a tedious and error-prone process.

Lastly, this abstraction is too low-level. It forces developers to study and measure the impact of individual operations, prohibiting its wide adoption in practice.

Application-specific optimizations do not generalize. Because each application has different performance metrics and relies on different features, a fine-tuned policy for one application will often work poorly for another. For example, DASH [185] optimizes QoE for video streaming; it keeps a high frame rate and reduces resolutions for adaptation. Its policy that lowers the resolution works poorly for video analytics that relies on image details [133, 203]. In Figure 3.3c, we show that pedestrian detection accuracy drops fast when reducing resolutions as pedestrian are small in the scenes.

Similar applications face different data distributions, as shown in Figure 3.3 between stationary cameras detecting pedestrians (left) and mobile cameras recognizing objects (right). For stationary cameras, when we consider the slow walking speed of pedestrians, a high frame rate is not necessary. But high-resolution images are crucial because these surveillance cameras are far away from the targets. In the mobile camera case, because the camera moves, reducing the frame rate introduces significant errors.

3.3 AWStream Design

To address the issues with manual policies or application-specific optimizations, AWStream structures adaptation as a set of approximate, modular, and extensible specifications (§3.3.1). The well-defined structure allows us to build a generic profiling tool that learns an accurate relationship—we call it the profile—between bandwidth consumption and application accuracy (§3.3.2). The profile then guides the runtime to react with precision: achieving low latency and high accuracy when facing insufficient bandwidth (§3.3.3). Figure 3.4 shows the high-level overview of AWStream.

3.3.1 API for Structured Adaptation

Most stream processing systems construct applications as a directed graph of operators [201, 218]. Each operator transforms input streams into new streams. AWStream borrows the same computation

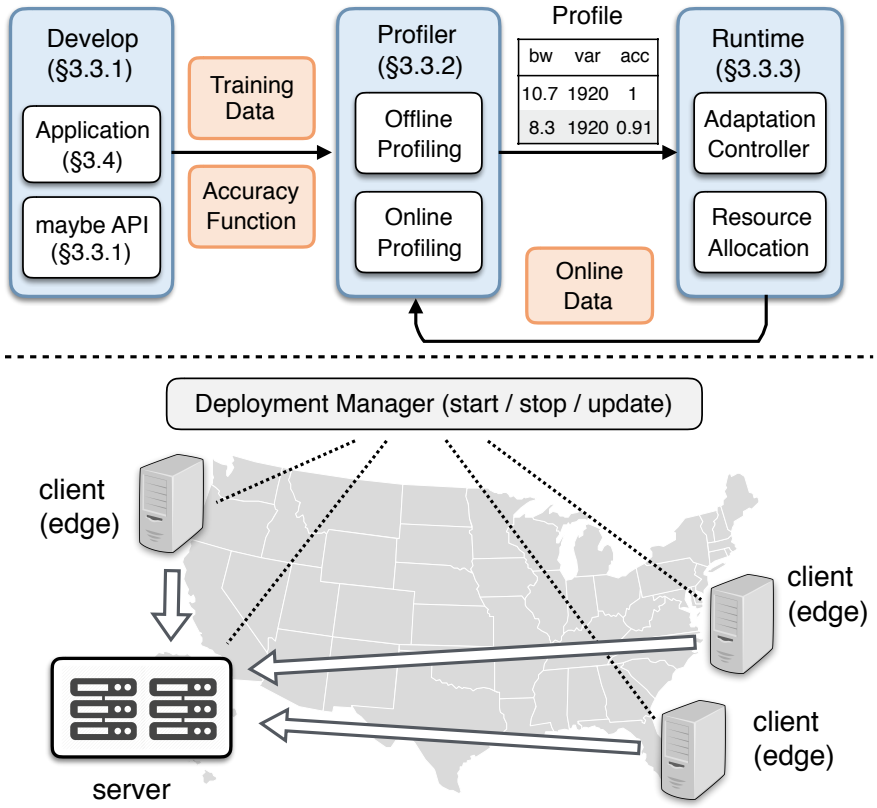


Figure 3.4: AWStream's phases: development, profiling, and runtime. AWStream also manages wide-area deployment.

Normal Operators	*map* (f: I ⇒ O)	Stream<I> ⇒ Stream<O>
	skip (i: Int)	Stream<I> ⇒ Stream<I>
	sliding_window (count: Int, f: Vec<I> ⇒ O)	Stream<I> ⇒ Stream<O>
	tumbling_window (count: Int, 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<Int>)	Stream<I> ⇒ Stream<I>
	maybe_head (knobs: Vec<Int>)	Stream<Vec<I>> ⇒ Stream<Vec<I>>
	maybe_downsample (knobs: Vec<(Int, Int)>)	Stream<Image> ⇒ Stream<Image>
	...	...

Table 3.1: Stream processing operators in AWStream. `Vec<T>` represents a list of elements with type `T`.

model and can support normal operators found in existing stream processing systems such as JetStream [160] (see example operators in Table 3.1).

To integrate adaptation as a first-class abstraction, AWStream introduces `maybe` operators that degrade data quality, yielding potential bandwidth savings. Our API design has three considerations. (i) To free developers from specifying exact rules, the API should allow specifications with options. (ii) To allow combining multiple dimensions, the API should be modular. (iii) To support flexible integration with arbitrary degradation functions, the API should take user-defined functions. Therefore, our API is,

```
maybe(knobs: Vec<T>, f: (T, I) => I)
```

We illustrate the use of the `maybe` operator with an example that quantizes a stream of integers in Rust:

```
let quantized_stream = vec![1, 2, 3, 4].into_stream()
    .maybe(vec![2, 4], |k, val| val.wrapping_div(k))
    .collect();
```

The snippet creates a stream of integers, chains a degradation operation, and collects the execution result. In this example, the knob is [2, 4] and the degradation function performs a wrapping (modular) division where the divisor is the chosen knob. The knob value modifies the quantization level, affecting the output: [1, 2, 3, 4] (no degradation), [0, 1, 1, 2] (k=2), or [0, 0, 0, 1] (k=4). If the stream is then encoded—for example, run-length encoding as in JPEG [209]—for transmission, the data size will depend on the level of degradation.

Based on the `maybe` primitive, one can implement additional degradation operators for common data types. For instance, `maybe_head` will optionally take the top values of a list; `maybe_downsample` can resize the image to a configured resolution. AWStream provides a number of such operations as a library to simplify application development (Table 3.1).

With our API, the example mentioned in §3.2.3 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 instantiates a `Camera` source, which produces `Stream<Image>` with 1920x1080 resolution and 30 FPS. Two degradation operations follow the source: one that downsamples the image to 1600x900 or 1280x720 resolution, and the other that skips every 2 or 5 frames, resulting in 30/(2+1)=10 FPS or 30/(5+1)= 6 FPS. This example then displays degraded images. In practice, operators for further processing, such as encoding and transmission, can be chained.

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-optimal set
c_i, c_{i+1}, $c_{\max}$	current/next/maximal configuration at runtime
R	network delivery rate (estimated bandwidth)
$\mathrm{Q_E}$, $\mathrm{Q_C}$	messages when `Queue` is empty or congested
$\mathrm{R_C}$	message when `Receiver` detects congestion
$\mathrm{AC_{Probe}}$	message when `AC` requests probing
$\mathrm{S_{ProbeDone}}$	message when `Socket` finishes probing

Table 3.2: Notations used in this chapter.

3.3.2 Automatic Profiling

After developers use `maybe` operators to specify potential degradation operations, AWStream automatically builds an accurate profile. The profile captures the relationship between *application accuracy* and *bandwidth consumption* under different combinations of data degradation operations. We describe the formalism, followed by techniques that efficiently perform offline and online profiling.

Profiling formalism. Suppose a stream processing application has n `maybe` operators. Each operator introduces a knob k_i. The combination of all knobs forms a *configuration* $c = [k_1, k_2, ...k_n]$. The set of all possible configurations $\mathbb{C}$ is the space that the profiling explores. For each configuration c, there are two mappings that are of particular interest: a mapping from c to its bandwidth consumption $B(c)$ and its accuracy measure $A(c)$. Table 3.2 summarizes these symbols.

The profiling looks for Pareto-optimal configurations; that is, for any configuration c in the Pareto-optimal set $\mathbb{P}$, there is no alternative configuration c' that requires less bandwidth and offers a higher accuracy. Formally, $\mathbb{P}$ is defined as follows:

$$\mathbb{P} = \{c \in \mathbb{C} : \{c' \in \mathbb{C} : B(c') < B(c), A(c') > A(c)\} = \varnothing\} \tag{3.1}$$

We show examples of knobs, configurations, and accuracy functions when we present applications in §3.4 and visualize the profile of sample applications in Figure 3.10.

Offline Profiling. We first use an offline process to build a bootstrap profile (or default profile). Because AWStream allows arbitrary functions as the degradation functions, it does not assume a closed-form relationship for $B(c)$ and $A(c)$. AWStream takes a data-driven approach: profiling

applications with developer-supplied training data. $B(c)$ is measured as the data rate (bps) at the point of transmission. The accuracy $A(c)$ is measured either against the groundtruth, or the reference results when all degradation operations are off.

AWStream makes no assumptions on the performance models, and thus evaluates all possible configurations. While all knobs form a combinatorial space, the offline profiling is only a one-time process. We exploit parallelism to reduce the profiling time. Without any *a priori* knowledge, all configurations are assigned randomly to available machines.

Online Profiling: AWStream supports online profiling to continuously refine the profile. The refinement handles *model drift*, a problem when the learned profile fails to predict the performance accurately. There are two challenges with online profiling. (i) There are no ground-truth labels or reference data to compute accuracy. Because labeling data is prohibitively labor intensive and time consuming [167], AWStream currently uses raw data (data without degradation) as the reference. At runtime, if the application streams raw data, it is used for online profiling. Otherwise, we allocate additional bandwidth to transmit raw data, but only do so when there is spare capacity. (ii) Exhaustive profiling is expensive. If the profiling takes too much time, the newly-learned profile may already be stale. AWStream uses a combination of parallelization and sampling to speed up profiling, as below:

- Parallelization with degradation-aware scheduling. Evaluating each configuration takes a different amount of time. Typically, an increase in the level of degradation leads to a decrease in computation; for example, a smaller FPS means fewer images to process. Therefore, we collect processing times for each configuration from offline profiling and schedule online profiling with longest first schedule (LFS) [110] during parallelization.
- Sampling-based profiling. Online profiling can speed up when we sample data or configurations. Sampling data reduces the amount of data to process, but at a cost of generating a less accurate profile. When sampling configuration, we can evaluate a subset of the Pareto-optimal configurations and compare their performances with an existing profile. A substantial difference, such as more than 1 Mbps of bandwidth estimation, triggers a full profiling over all configurations to update the current profile.

3.3.3 Runtime Adaptation

At runtime, AWStream matches data rate to available bandwidth to minimize latency and uses Pareto-optimal configurations to maximize accuracy. This section focuses on the details of our runtime design. We defer the evaluation and comparisons with existing systems (e.g., JetStream) to §3.5.3.

Figure 3.5 shows our runtime system architecture. AWStream applications' source contains a `Maybe` module derived from all `maybe` operators. This module allows the controller to update the level of degradation. Data generated by the source is then enqueued to `Queue` and subsequently dequeued by `Socket`, which sends data over the network. `Socket` uses TCP as the underly-

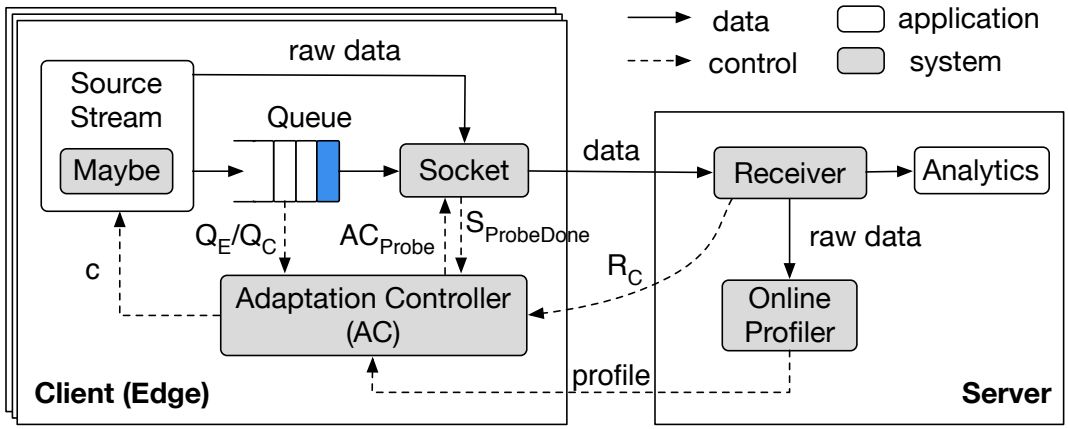
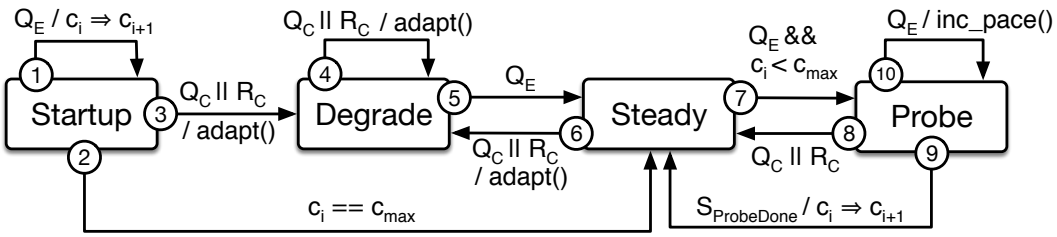


Figure 3.5: Runtime adaptation system architecture.

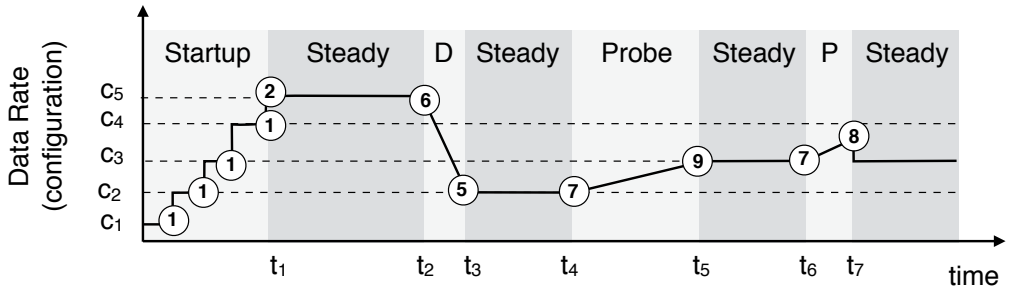
ing protocol for congestion control and estimates the available bandwidth using application-level delivery rate. When the data generation rate exceeds `Socket`'s departure rate, the queue grows. In this case, the adaptation controller (AC) queries the estimated bandwidth from `Socket` and regulates the source stream by updating the configuration. After the data is sent through the network, `Receiver` delivers data to the application analytics. `Receiver` also performs congestion detection and extracts raw data, if it is present. It tracks the minimal latency (similar to how BBR tracks `RTprop` [43]) and reports sudden application-level latency spikes to clients as congestion signals (R_C). If a new profile is learned by the online profiler, it is fed back to AC for subsequent adaptation.

Figure 3.6a shows the adaptation algorithm with a state machine model and Figure 3.6b shows the state transitions with an example. We first describe all symbols. AC loads the profile and sorts all configurations with an ascending order of bandwidth demand, resulting in a list $[c_1, \ldots, c_{\max}]$. These configurations follow a total order: $c_i < c_j$ if $B(c_i) < B(c_j)$. We denote the current configuration as c_i and the next c_{i+1}. AC receives messages from other modules: Q_E when `Queue` is empty; Q_C when queued items exceed a threshold; and R_C when `Receiver` detects congestion. AC can query `Socket` for delivery rate R (arrow not shown) or request it to probe (AC_{Probe}) for a target bandwidth, often $B(c_{i+1})$. If there is no congestion during the probing and $R > B(c_{i+1})$, `Socket` sends back $S_{ProbeDone}$. Below, we describe each state and transitions.

- **Startup: rapid growth.** AWStream starts with c_1 and grows the rate ($c_i \Rightarrow c_{i+1}$) upon each Q_E. The growth stops at $c_{\max}$ (to `Steady`) or if it receives Q_C/R_C (to `Degrade`).
- **Degrade: reacting to congestion.** Congestion is detected in two ways: (1) when `Queue` grows and exceeds a threshold, AC receives Q_C; (2) when `Receiver` detects latency spikes, AC receives R_C. During congestion, AC runs the `adapt()` procedure by updating `Maybe` with the maximum-allowed c that satisfies $B(c) < \alpha R$, where $\alpha \in (0, 1)$ and R is `Socket`'s current delivery rate. A smaller α allows a quicker draining of the queue. After the congestion is resolved (Q_E received), AWStream changes to `Steady`.



(a) Rate adaptation as a state machine.



(b) An example illustrating the adaptation algorithm.

Figure 3.6: Runtime adaptation algorithm.

- **Steady: low latency delivery.** AWStream achieves low latency by spending most of the time in `Steady`. It changes to `Degrade` when congestion occurs. If $c < c_{\max}$ and it receives Q_E, AC starts `Probe` to check for more available bandwidth.
- **Probe: more bandwidth for a higher accuracy.** Advancing c_i directly may cause congestion if $B(c_{i+1}) \gg B(c_i)$. To allow a smooth increase, AC requests `Socket` to probe by sending additional traffic controlled by `probe_gain` (in `inc_pace()`, similar to BBR [43]). Raw data is used for probing if available, otherwise we inject dummy traffic. AWStream stops probing under two conditions: (1) upon $S_{\text{ProbeDone}}$, it advances c_i; (2) upon Q_C or R_C, it returns to `Steady`. The explicit `Probe` phase stabilizes feedback loop and prevents oscillation.

3.3.4 Resource Allocation & Fairness

In addition to rate adaptation, the profile is also useful for controlling a single application's bandwidth usage or allocating resources among competing tasks.

For individual applications, developers can pin-point a configuration for a given bandwidth or accuracy goal. They can also specify a criterion to limit effective configurations. For example, AWStream can enforce an upper bound on the bandwidth consumption (e.g., do not exceed 1 Mbps) or a lower bound on application accuracy (e.g., do not fall below 75%).

For multiple applications, their profiles allow novel bandwidth allocation schemes such as utility fairness. Different from resource fairness with which applications get an equal share of bandwidth,

Application	Knobs	Accuracy	Dataset
Augmented Reality	resolution frame rate quantization	F1 score [166]	iPhone video clips training: office (24 s) testing: home (246 s)
Pedestrian Detection	resolution frame rate quantization	F1 score	MOT16 [142] training: MOT16-04 testing: MOT16-03
Log Analysis (Top-K, K=50)	head (N) threshold (T)	Kendall's τ [4]	SEC.gov logs [64] training: 4 days testing: 16 days

Table 3.3: Application details.

utility fairness aims to maximize the *minimal* application accuracy. With the profiles, bandwidth allocation is equivalent to finding proper configuration c^t for application t. We formulate utility fairness as follows:

$$\begin{aligned} \underset{c^t}{\text{maximize}} \quad & \min(A^t(c^t)) \\ \text{subject to} \quad & \sum_t B^t(c^t) < R \end{aligned} \tag{3.2}$$

Solving this optimization is computationally hard. AWStream uses heuristics that are similar to VideoStorm [224]: it starts with c_1^t and improves the application t with the worst accuracy; this process iterates until all bandwidth is allocated. In this thesis, we demonstrate resource allocation with two applications as a proof-of-concept in §3.5.4.

3.4 Implementation

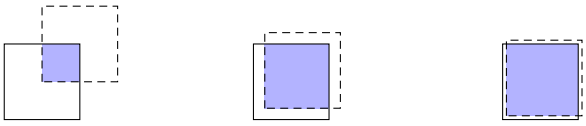
While our proposed API is general and not language specific, we have implemented AWStream prototype in Rust (~4000 lines of code). AWStream is open source on GitHub.[3] Applications use AWStream as a library and configure the execution mode—profiling, runtime as client, or runtime as server—with command line arguments.

Using AWStream, we have built three applications: augmented reality (AR) that recognizes nearby objects on mobile phones, pedestrian detection (PD) for surveillance cameras, and a distributed log analysis to extract the Top-K mostly accessed files (TK). Table 3.3 summarizes the application-specific parts: knobs, accuracy functions, and datasets.

[3] https://github.com/awstream/awstream

Figure 3.7: Three AWStream applications: augmented reality, pedestrian detection, and distributed Top-K.

$$\text{IOU} = \frac{\text{Area of Intersection}}{\text{Area of Union}}$$



(a) IOU=0.14 **(b)** IOU=0.57 **(c)** IOU=0.82

Figure 3.8: Intersection over Union (IOU) illustration.

Augmented Reality. We target at augmented reality applications running on mobile phones that recognize objects by offloading the heavy computation elsewhere (e.g., the cloud).

Our implementation uses OpenCV [37] for image-related operations and YOLO [162, 163], a GPU-enabled pre-trained neural network, for object recognition. Videos are encoded with H.264 [165]. Our implementation uses GStreamer [194] with `x264enc` plugin (`zerolatency` and constant quality). The quantization factor affecting encoding quality becomes a knob in addition to image resolutions and frame rates.

Object recognition returns a list of bounding boxes with the object type. Each bounding box is a rectangle with normalized coordinates on the image. We compare the detection against the reference result from raw data, and declare it success if the intersection over union (IOU) is greater than 50% [76] (illustrated in Figure 3.8) and the object type matches. We use F1 score [166] as the accuracy function. In terms of dataset, we collected our own video clips: the training data is a 24-second long video of an office environment; the test data is a 246-second long video of a home environment.

Pedestrian Detection. This application analyzes streams of videos from installed CCTV cameras and detects pedestrians inside. We use a similar setup (OpenCV and GStreamer) as our augmented reality application except for the analytical function. To detect pedestrians, we use GPU-accelerated histogram of oriented gradients (HOG) [59] with the default linear SVM classifier from OpenCV. Because we do not recognize individual pedestrians, a successful detection in this case only requires matching the bounding box. Our evaluation uses MOT16 dataset [142] for both profiling and runtime.

Distributed Top-K. This application aggregates machine logs from geo-distributed servers to find out the Top-K most accessed files, similar to many Top-K queries [25].

Figure 3.9 illustrates our processing pipeline with two degradation operations. First each source

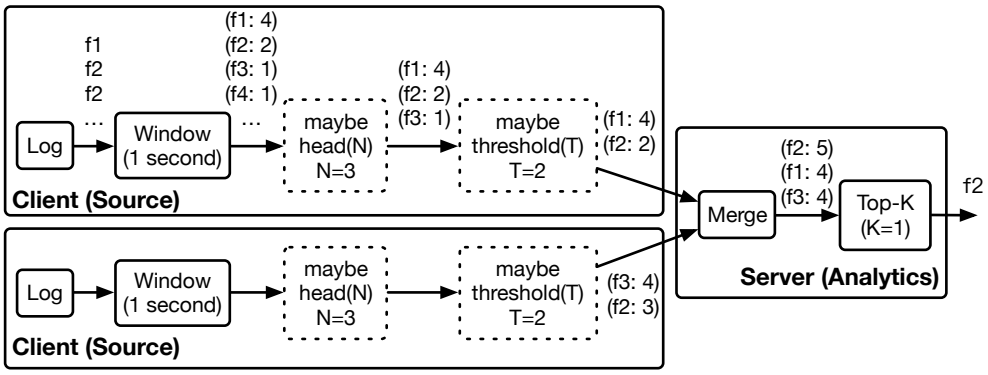


Figure 3.9: A distributed Top-K application with two degradation operations: `head` and `threshold`. In this example, `f2`, which is not in Top-1 for either client, becomes the global Top-1 after the merge. It would have been purged if the clients use threshold T=3, demonstrating degradation that reduces data sizes affects fidelity.

node summarizes the log using `Window` operator to reduce the data size, a pre-processing step. As many real-world access patterns follow a long tail distribution, there can be a large-but-irrelevant tail that contributes little to the final Top-K. Each source node then filters the tail: (1) head(`N`) takes the top `N` entries; (2) threshold(`T`) filters small entries whose count is smaller than `T`. These two operations affect the final result and the exact impact depends on data distribution. We implement these two operators by using AWStream's `maybe` abstraction.

To measure the accuracy, we need to compare the correlation between two ranked list. Kendall's τ [4] is a correlation measure of the concordance between two ranked list. The output ranges from -1 to 1, representing no agreement to complete agreement. To integrate with AWStream, we convert Kendall's τ to [0, 1] with a linear transformation. For our evaluation, we set K as 50 and use Apache log files that record and store user access statistics for the SEC.gov website. The logs are split into four groups, simulating four geo-distributed nodes monitoring web accesses. To match the load of popular web servers, we compress one hour's logs into one second.

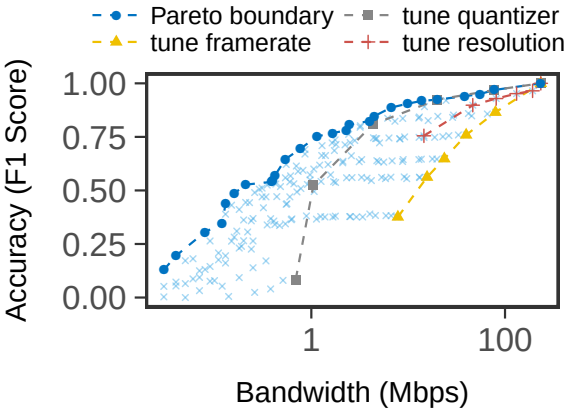
3.5 Evaluation

In this section, we show the evaluations of AWStream, summarizing the results as follows.

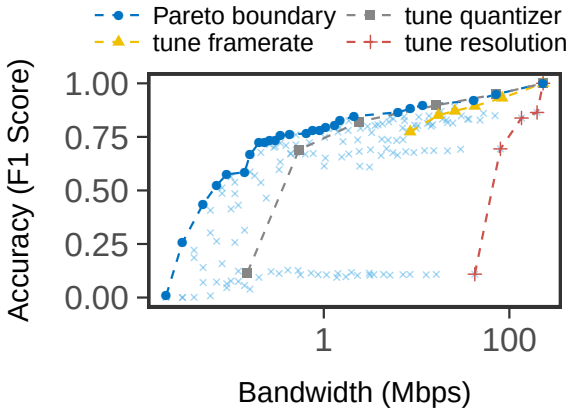
§3.5.1 AWStream generates Pareto-optimal profiles across multiple dimensions with precision (Figure 3.10).

§3.5.2 Our parallel and sampling techniques speeds up offline and online profiling (Figure 3.11, Figure 3.12).

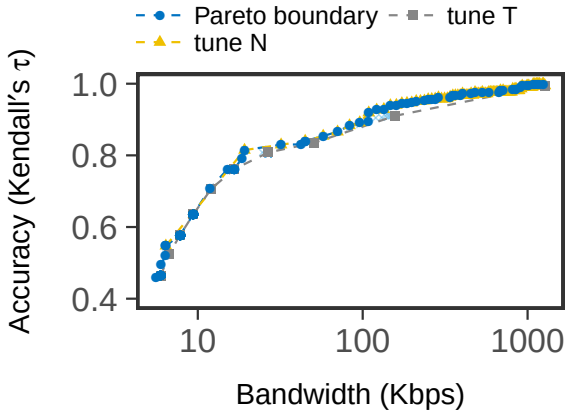
§3.5.3 At runtime, AWStream achieves sub-second latency and nominal accuracy drop for all applications (Figure 3.13, Figure 3.14) and across various network conditions (Figure 3.15).



(a) Augmented Reality (AR)



(b) Pedestrian Detection (PD)



(c) Top-K (TK)

Figure 3.10: Application profiles of three applications. Each cross point is one configuration c's performance $(B(c), A(c))$. All figures show the Pareto boundary as well as the performance if only tuning one dimension. Note the x-axis is in log scale.

§3.5.4 AWStream profiles allow different resource allocations: resource fairness and utility fairness (Figure 3.16).

3.5.1 Application Profiles

We run offline profiling using the training dataset described in Table 3.3 and show the learned profiles in Figure 3.10. In each figure, the cross dots represent the bandwidth demand and application accuracy for one configuration. We highlight the Pareto-optimal boundary $\mathbb{P}$ with blue dashed lines. To understand each dimension's impact on the degradation, we highlight configurations from tuning only *one* dimension. From these profiles, we make the following observations:

Large bandwidth variation. For all three applications, The bandwidth requirements of all three applications have two to three orders of magnitude of difference (note the x-axis is in log scale).

For AR and PD, the most expensive configuration transmits videos at 1920x1080, 30 FPS and 0 quantization; it consumes 230 Mbps. In contrast to the large bandwidth variation, there is a smaller variation in accuracy. In PD, for example, even after the bandwidth reduces to 1 Mbps (less than 1% of the maximum), the accuracy is still above 75%. The large variation allows AWStream to operate at a high accuracy configuration even under severe network deterioration.

Distinct effects by each dimension. Comparing dashed lines in each profile, we see that the Pareto-optimal configurations are only achievable when multiple knobs are in effect. Tuning only one dimension often leads to sub-optimal performance. Within a single profile, the difference between tuning individual dimensions is evident. For PD, tuning resolution (the red line) leads to a quicker accuracy drop than tuning frame rate (the yellow line). Comparing AR and PD, the same dimension has different impact. Tuning resolution is less harmful in AR than PD; while tuning frame rate hurts AR more than PD. This echoes our initial observation in §3.2.3 that application-specific optimizations do not generalize.

Quantification with precision. The generated profiles are actionable configurations that control the knobs with precision. For example, if PD transmits video at 1920x1080 resolution, 10 FPS and a quantization of 20, it will consume 11.7 mbps of bandwidth, achieving roughly 90% accuracy. This saves developers from laboriously analyzing their application to compute manual policies.

3.5.2 Profiling Efficiency & Online Profiling

This section focuses on the AR application as a case study; our profiling techniques—parallelism and sampling—do not make assumptions about the application; therefore, the evaluation results can be generalized to other applications.

In AR, there are 216 different configurations: 6 resolutions, 6 frame rates and 6 quantization levels. AR uses YOLO [163], a neural network model for object detection. It takes roughly 30 ms to process one frame on GeForce® GTX 970.[4] But different configurations require different times for processing. For example, a 10 FPS video has 1/3 of the frames to process in comparison to a 30 FPS video. In our experiment, to evaluate all 216 configurations, it takes 52 seconds for 1 second worth of data. We denote such overhead as 52X. Section 3.2 discusses parallel and sampling techniques to improve the profiling efficiency; we present their evaluations as follows.

Parallelism reduces the profiling time (Figure 3.11). Because evaluating each individual configuration is independent of other configurations, we parallelize the profiling task by assigning configurations to GPUs. (i) Our offline profiling assigns configurations randomly. With the increased number of GPUs, the overhead reduces from 52X to 4X with 30 GPUs. (ii) Our online profiling assigns configurations based on the processing times collected during offline. AWStream uses LFS [110] to minimize the makespan and reduces the overhead to 1.75X with 30 GPUs (29× gain).

[4] YOLO resizes images to fixed 416×416 resolutions as required by the neural network. Evaluating images with different resolutions takes similar time.

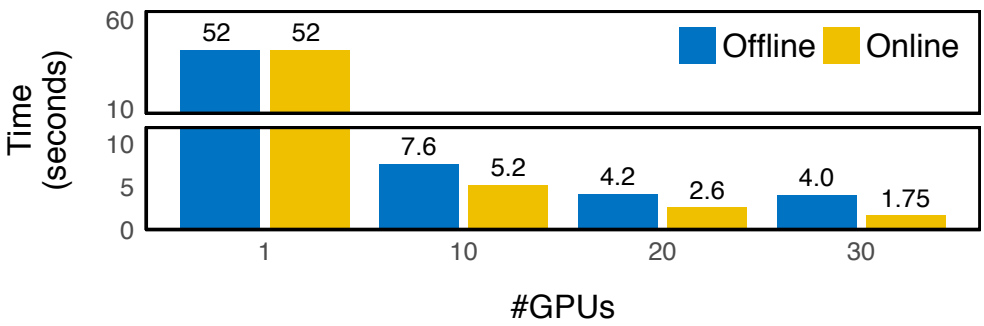


Figure 3.11: Parallelism speeds up both offline and online profiling. The y-axis shows the profiling time for 1-second video.

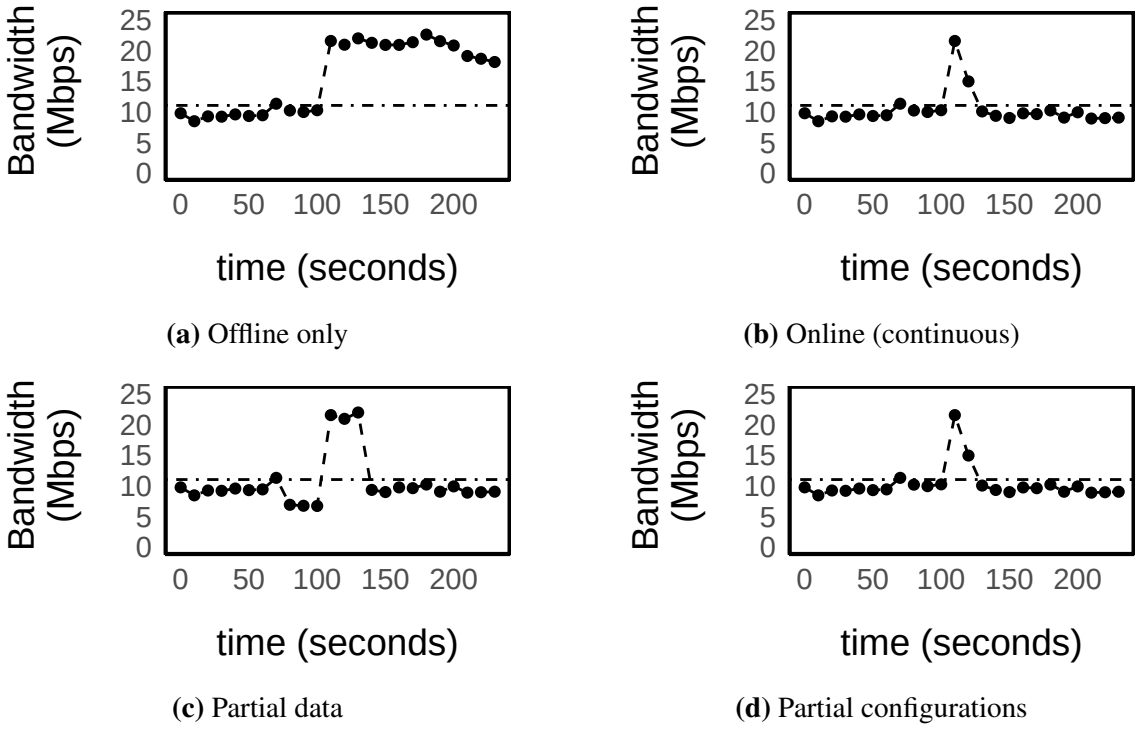
Sampling techniques speed up online profiling (Figure 3.12). Before we evaluate the speed up, we validate *model drift* with real-world data. When using the profile trained in an office environment, the application should use a configuration of 1280x720, 30 FPS and 20 quantization to meet an 11 Mbps goal. We test it against a home environment; but at about t=100s, the camera points out of the window to detect objects on the street. Because of the scene change, the configuration fails to predict bandwidth, as illustrated in Figure 3.12a.

To correct the profile, if we continuously run the profiling online and update the profile, the application will choose the right configuration to meet the bandwidth limit. Figure 3.12b shows the bandwidth prediction when we continuously profile with the past 30 seconds of video. At time t=120s, the new prediction corrects the drift. The downside of continuous profiling, as discussed earlier, is the cost: 52X overhead with 1 GPU. In addition to parallelism, AWStream uses sampling techniques for online profiling (improvements in Table 3.4):

(i) Partial data. Instead of using all the past data, we run profiling with only a fraction of the raw data. Figure 3.12c shows the bandwidth consumption if the profiling uses only 10 seconds of data out of the past 30 seconds. In this way, although the profile may be less accurate (the mis-prediction at t=80-100s), and there is a delay in reacting to data change (the mis-prediction is corrected after t=125s), we save the online profiling by 3× (from 52X to 17X).

(ii) Partial configurations. If we use the past profile as a reference and only measure a subset of $\mathbb{P}$, the savings can be substantial. A full profiling is only triggered if there is a significant difference. Figure 3.12d shows the bandwidth prediction if we evaluate 5 configurations continuously and trigger a full profiling when the bandwidth estimation is off by 1 Mbps or the accuracy is off by 10%. For our test data, this scheme is enough to correct model drifts by predicting an accurate bandwidth usage (compare Figure 3.12b and Figure 3.12d). The overhead reduces to 6X because we run full profiling less often (only two full profiling). It is an 8.7× gain.

Note that these techniques—parallelization, sampling data, and sampling configurations—can be combined to further reduce the profiling overhead. For example, scheduling 5 GPUs running 5 configurations continuously to check for model drift will reduce the overhead to 1X. In practice, the amount of resources to use depends on the budget and the importance of the job. AWStream currently requires developers to configure the application with proper online profiling techniques.



(a) Offline only

(b) Online (continuous)

(c) Partial data

(d) Partial configurations

Figure 3.12: The horizontal reference line is the target bandwidth (11 Mbps). (1) Online profiling is necessary to handle model drift ((a) vs. (b-d)). (2) Sampling techniques—partial data (c) and partial configurations (d)—can correct model drift with less profiling overhead (see Table 3.4), compared to continuous (b). We omit accuracy predictions since in all schemes AWStream finds configurations that achieve similarly high accuracy (~90%).

Online scheme	Overhead	Improvements
Continuous	52X	Baseline
Partial data	17X	3×
Partial configurations	6X	8.7×

Table 3.4: Compared to the continuous profiling baseline (52X overhead), our sampling techniques speed up by 3× or 8.7×.

3.5.3 Runtime Adaptation

In this section, we evaluate the runtime performance by controlling bandwidth across geo-distributed sites and compare AWStream with baselines including streaming over TCP/UDP, JetStream, and video streaming. Due to limited space, we discuss AR in depth and only present the results of PD/TK.

Experiment setup. We conduct our experiments on four geo-distributed machines from Amazon

EC2, spanning four different regions. Three (at N. Virginia, Ohio, Oregon) act as worker nodes and one (at N. California) acts as the analytics server. The average RTTs from the workers to the server are 65.2 ms, 22.2 ms, and 50.3 ms.

During the experiment, each worker transmits test data (Table 3.3) for about 10 mins. If the duration of the test data is less than 10 mins, it loops. Because $B(c_{\max})$ is prohibitively large (raw videos consumes 230 Mbps), we use a reasonable configuration to limit the maximum rate. In our AR experiment, $c_{\max}$ is 1600x900 resolution, 30 FPS and 20 quantization; it consumes about 14 Mbps.

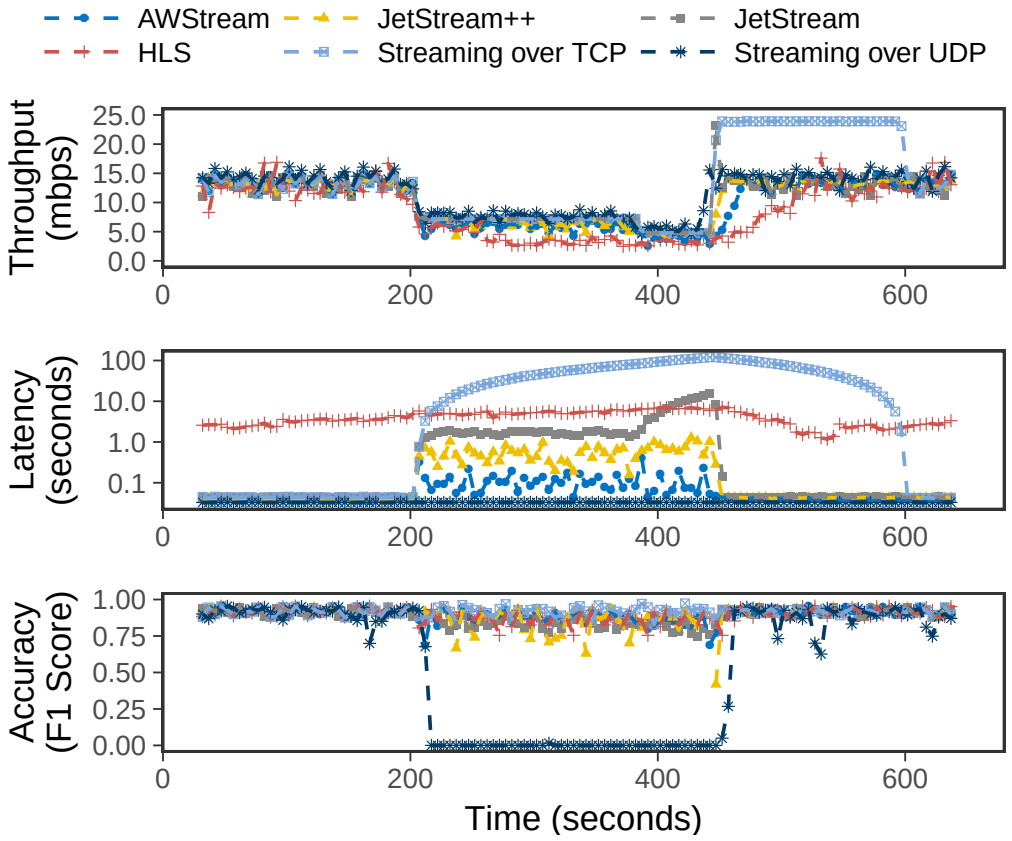
Our bandwidth control scheme follows JetStream [160]. During the experiment, we use the Linux `tc` utility with HTB [66, 104] to control the clients' outgoing bandwidth. Each experiment involves four phases: (i) before t=200s, there is no shaping; (ii) at t=200s, we limit the bandwidth to 7.5 Mbps for 3 minutes; (iii) at t=380s, we further decrease the bandwidth to 5 Mbps; (iv) at t=440s, we remove all traffic shaping. For UDP, HTB doesn't emulate the packet loss or out-of-order delivery; so we use `netem` and configure the loss probability according to the delivery rate. Because each pair-wise connection has a different capacity, we impose a *background* bandwidth limit—25 Mbps—such that all clients can use at most 25 Mbps of network bandwidth.

We compare AWStream with the following baselines:

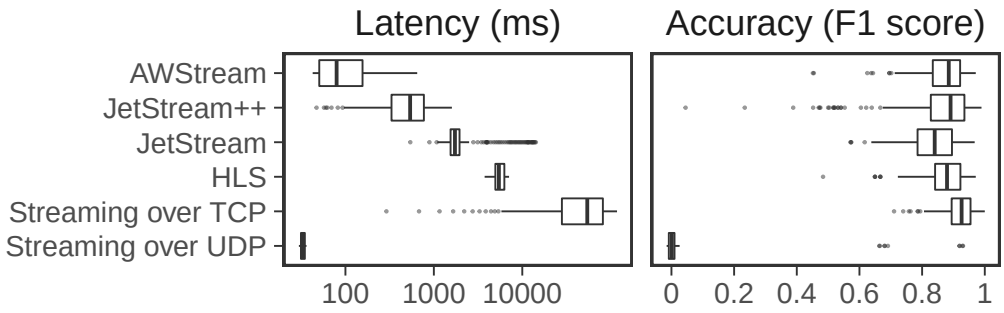
- Streaming over TCP/UDP (non-adaptive). For TCP, we re-use AWStream runtime that runs over TCP but disable the adaptation. For UDP, we use FFmpeg [29] to stream video: RTP/UDP [176] for media and RTSP for signaling [175]; as in typical video conferencing and IP cameras [71, 114].
- Adaptive video streaming. We use HTTP Live Streaming (HLS) to represent popular adaptive video streaming techniques. Our setup resembles personalized live streaming systems [210] but uses a smaller chunk for low latency (1 second instead of typical 2-10 seconds). Additional information about the setup for HLS is available in §3.5.5.
- JetStream with the manual policy described in §3.2.3.
- JetStream++, a modified version of JetStream that uses the profile learned by AWStream. Additional information about the setup for HLS is available in §3.5.5.

At runtime, AWStream differs from JetStream in both policy and adaptation. JetStream++ improves over JetStream by using our Pareto-optimal profile. AWStream improves the performance further with two major changes: (i) AWStream directly measures the delivery rate to select an appropriate configuration to match available bandwidth while JetStream employs a latency-based measure of capacity ratio; (ii) AWStream has an explicit probe phase while JetStream changes its policy immediately after capacity ratio changes.

Results. Figure 3.13a shows the runtime behavior of AWStream and all baselines in time series. Figure 3.13b summarizes the latency and accuracy with box plots during bandwidth shaping (between t=200s and t=440s).



(a) Time-series plot of the runtime behaviors: throughput (top), showing the effect of bandwidth shaping; latency (middle) in log scale; and accuracy (bottom). Overlapped lines may be hard to read; we present the same results in Figure 3.13b for clarity.



(b) Latency and accuracy during the traffic shaping (t=200s–440s).

Figure 3.13: For AR, AWStream simultaneously achieves low latency and high accuracy (accuracy has a smaller variation in comparison with other baselines).

The throughput figure (Figure 3.13a) shows the effect of traffic shaping. During the shaping, TCP and UDP make full use of the available bandwidth; in comparison, AWStream, JetStream, JetStream++, and HLS are conservative because of adaptation (see their throughput drops). When we stop shaping at t=440s, TCP catches up by sending all queued items as fast as possible. JetStream also has queued items because the policy in use (with only three rules) cannot sustain 5 Mpbs bandwidth. AWStream's throughput increases gradually due to the explicit probing phase. HLS is the most conservative scheme; it does not recover from degradation until t=500s.

The latency figures (both Figure 3.13a and Figure 3.13b) show that AWStream is able to maintain sub-second latency. During the traffic shaping, TCP queues items at the sender side for up to hundreds of seconds. In contrast, UDP always transmits as fast as possible, leading to a consistent low latency.[5] HLS's latency fluctuates around 4-5 seconds due to chunking, buffering, and network variations, on par with recent literature [210]. Both JetStream and JetStream++ are able to adapt during traffic shaping. With a more precise and fine-grain policy, JetStream++ achieves a lower latency (median 539 ms) in comparison to JetStream (median 1732 ms). Because JetStream's runtime reacts instantaneously when the congestion condition changes, both baselines easily overcompensate and exhibit oscillation among polices during the experiment. AWStream effectively addresses the oscillation with probing and achieves a much lower latency: median 118 ms, 15× improvement over JetStream and 5× improvement over JetStream++.

The accuracy figures (both Figure 3.13a and Figure 3.13b) show that other than UDP, most schemes are able to maintain high accuracy. streaming over TCP always sends data at high fidelity, achieving the highest accuracy (median 93%), but at a cost of high latency. JetStream uses a manual policy that are sub-optimal in comparison to our learned profile, so its accuracy is low (median 84%). Using Pareto-optimal configurations, JetStream++ is able to achieve a higher accuracy (median 89%); but because JetStream's runtime oscillates the policy, the accuracy has a large variation (standard deviation 14%). In contrast, AWStream chooses configurations carefully to stay in a steady state as much as possible. It achieves a high accuracy of 89% with a small variation (standard deviation 7.6%). HLS also achieves reasonable accuracy (median 87%) because its adaptation of tuning resolution and encoding quality is effective in AR. However, HLS's adaptation works poorly for PD (6% accuracy as in Figure 3.14a).

In summary, Figure 3.13 shows that AWStream achieves low latency and high accuracy simultaneously. The latency improvement over JetStream allows interactive applications, such as AR, to feel responsive rather than interrupted [149]. We show the results *during shaping* in a different form in Figure 3.1 to discuss the trade-off between fidelity and freshness.[6]

Pedestrian Detection. The setup for PD is the same with AR: three clients and one server on EC2. $c_{\max}$ is 1920x1080 resolution, 10 FPS and 20 quantization; it consumes about 12 Mbps. For PD, AWStream learns that resolution is more important than frame rate. Hence it favors 1080p with 10FPS over 900p with 30FPS. We use the same bandwidth shaping schedule and baselines as AR.

[5] FFmpeg discards packets that miss a deadline (33 ms for 30 FPS).

[6] We obtain Figure 3.1's app-specific data by feeding PD's profile to AR. We refer to JetStream as manual policies in Figure 3.1.

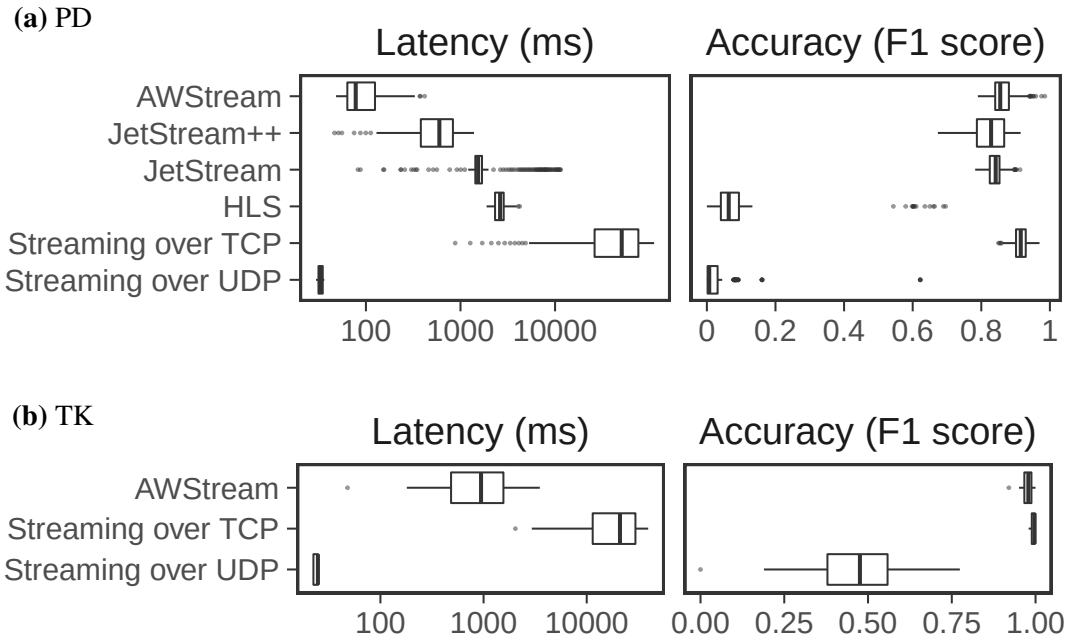


Figure 3.14: PD and TK performance summary. Box plot shows latency and accuracy during the traffic shaping (i.e., t=200s-440s).

Figure 3.14a shows the result and most observations about latency/accuracy are the same as AR. HLS has a poor accuracy because it reduces resolution and encoding quality during adaptation. AWStream is able to achieve the lowest latency (78 ms) with small accuracy drop (86%, 6% drop in comparison to TCP). In comparison to JetStream, AWStream improves the latency by 20× (from 1535 ms to 78 ms) and accuracy by 1% (from 84% to 85%).

Top-K. For TK, we use four clients and one server because our logs are split into four groups. $c_{\max}$ is $N = 9750$ for `head` and $T = 0$ for `threshold`; it consumes about 1.2 Mbps. Because the overall bandwidth consumption is much smaller than video analytics, we modify the bandwidth parameter: during t=200-380s, we limit the bandwidth to 750 Kbps; during t=380-440s, the bandwidth is 500 Kbps; the background limit is 2.5 Mpbs. We only compared AWStream with streaming over TCP and UDP. JetStream's Top-K is based on TPUT [41] that targets at queries asked hourly or daily. We did not implement our Top-K pipeline (Figure 3.9) with JetStream because video analytics suffice the purpose of comparison. Figure 3.14b shows the evaluation results. Streaming over TCP has the highest accuracy (99.7%) but the worst latency (up to 40 seconds). Streaming over UDP has the lowest latency but the worst accuracy (52%). AWStream achieves low latency (1.1 seconds) and high accuracy (98%) simultaneously. Notice that because TK's source generates data every second after `Window`, one object in the queue leads to one second latency.

Performance with Varying Network Delays

AWStream targets at wide area whose key characteristic is the large variation in latency [132].

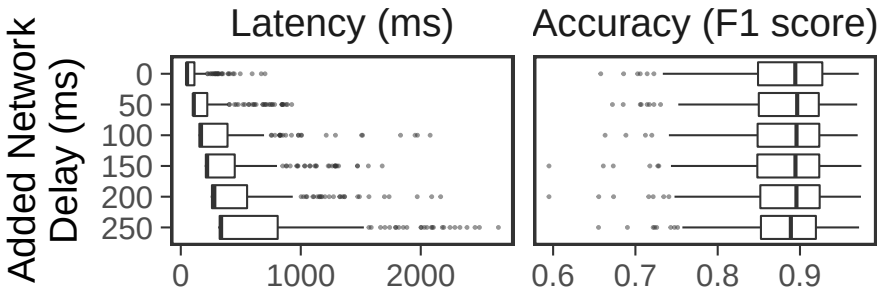
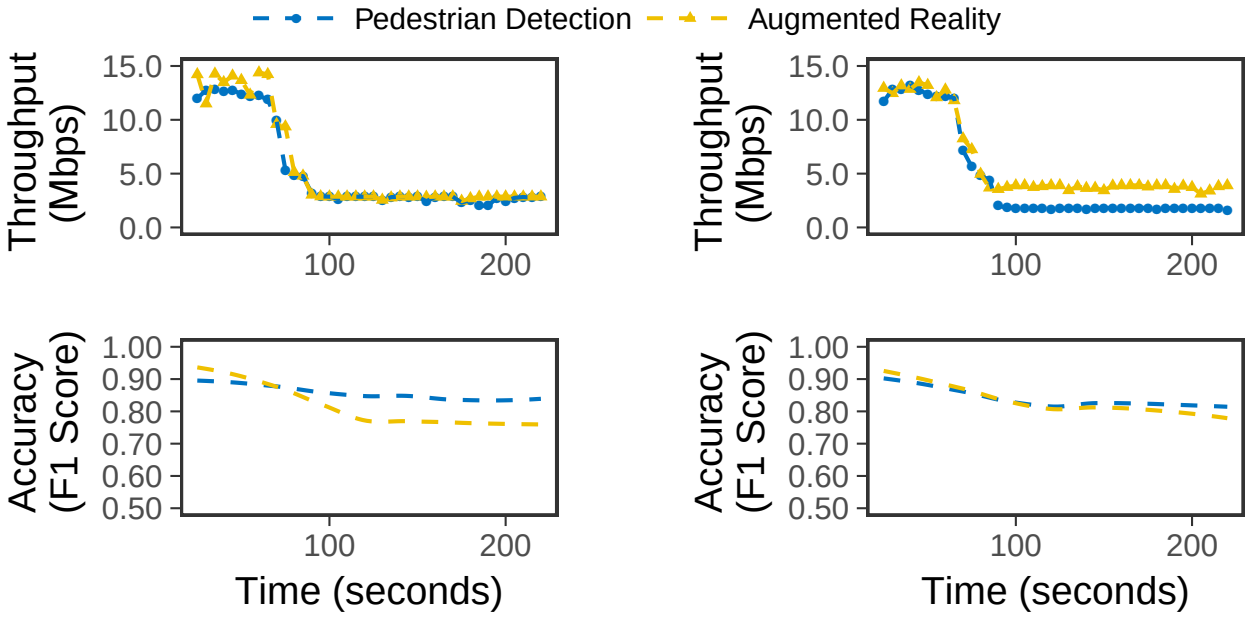


Figure 3.15: AWStream maintains low latency and high accuracy under different network delay conditions.



(a) Resource Fairness

(b) Utility Fairness

Figure 3.16: AWStream allows different resource allocation schemes.

While we have conducted experiments using real-world setup on EC2, the latency between EC2 sites is relatively low. To evaluate how AWStream performs with increased network delays, we conducted another experiment with one pair of client and server under different network conditions. We use `netem` to add delays, up to 250 ms each way, so the added RTT can be as high as 500 ms. The delay follows a normal distribution where the variation is 10% (e.g., 250 ± 25ms).

Figure 3.15 shows the runtime behavior with various added network delays. While the latency increases with the added delay, AWStream mostly manages to achieve sub-second latency for all conditions. We see a higher variation in latency and more outliers as network delay increases, because the congestion detection is slow when the RTT is high. In terms of accuracy, because AWStream mostly stays in `Steady` state and accuracy only depends on the level of degradation, AWStream achieves similar accuracy for different network delays.

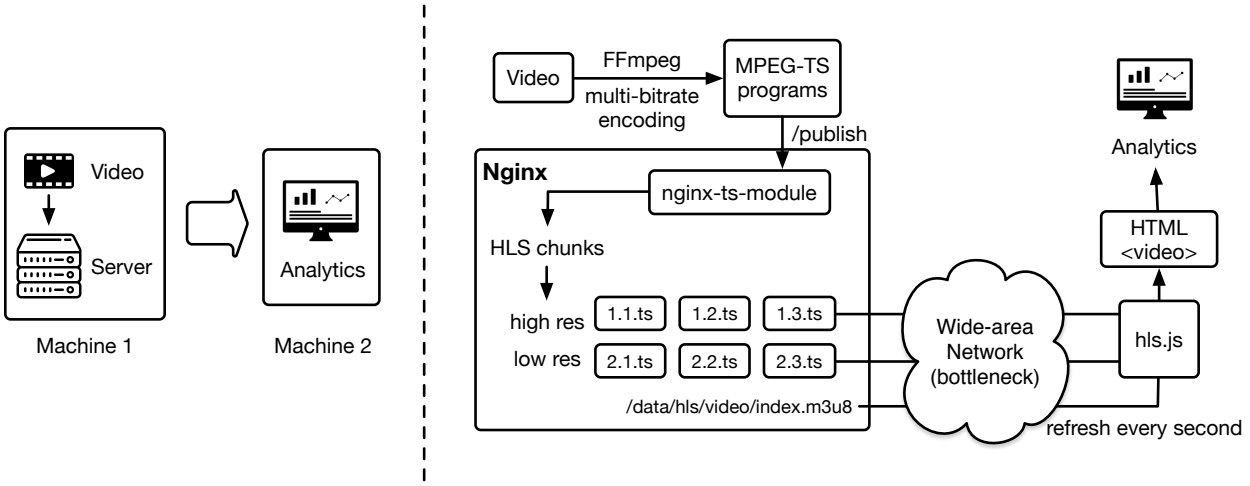


Figure 3.17: HLS setup. (Left) High-level overview: machine 1 generates a video and stores it in the server; machine 2 fetches the data and performs analytics. (Right) Detailed view: (1) FFmpeg encodes video with multiple bitrates and groups them into MPEG-TS programs; (2) `nginx-ts-module` then generates HLS chunks on the fly and stores them for nginx serving; (3) the client (using `hls.js`) periodically fetches the latest index file (`index.m3u8`) and then downloads the right chunk according to network conditions.

3.5.4 Resource Allocation and Fairness

We evaluate resource allocations with two applications. In this way, the result also covers the case of a single application, and can generalize to more applications.

We choose AR and PD as the example applications. The clients and servers of both applications are co-located so that they share the same bottleneck link. The experiment starts with sufficient bandwidth. At t=60s, we start traffic shaping to limit the total bandwidth to 6 Mbps. When we allocate resource equally between two applications (Figure 3.16a), each application gets 3 Mbps. Under this condition, PD runs with a higher accuracy of 85% while AR only achieves 77%. In addition to resource fairness, AWStream supports utility fairness: it chooses configurations that maximize the minimal accuracy. In this experiment, PD receives 2 Mbps and AR receives 4 Mbps; and both achieve 80% accuracy (Figure 3.16b).

3.5.5 HTTP Live Streaming

HTTP Live Streaming (HLS) [156] represents a class of HTTP-based media streaming protocols. Other protocols include Adobe HTTP Dynamic Streaming [5], Microsoft Smooth Streaming [219], and a newer vendor-independent standard DASH [138, 185]. These adaptive streaming protocols are widely adopted for both video-on-demand (VoD) and live streaming (such as Periscope).

Our setup (Figure 3.17) resembles the setup of popular live streaming services [210]. We first describe each component of our setup. We then discuss why these protocols are a poor match for wide-area streaming analytics, despite their adaptation capability.

Video. We use `FFmpeg` to encode the video with multiple bitrates: all levels use 30FPS, but different resolutions and H.264 encoding quality. Our experiment uses six different bitrates—900p, 720p, 540p, 360p, all with normal encoding quality; and 720p, 540p, with low encoding quality. To simulate live video streaming, we then use `FFmpeg` to re-publish the stream to an Nginx web server that accepts MPEG-TS (MPEG transport stream).

Server. Our web server is a nginx server compiled with plugin `nginx-ts-module` [22] that receives MPEG-TS over HTTP, produces live HLS and DASH chunks. It creates an index file `index.m3u8` that describes the media stream. During live streaming, the file is updated whenever new chunks arrive. And the client needs to fetch the newest version of `index.m3u8` to find out about new chunks. Typical streaming servers set each chunk to be 2-10 seconds [136, 190, 210]. For our low latency streaming, we configured the chunk segment to be 1 second.

Analytics. The HTTP client reads the index file and then fetches each chunk based on available bandwidth. Our client uses `hls.js` [65], a JavaScript library that implements an HLS client. It directly plays the video inside an HTML5 video element. To avoid invoking the graphical front-end of a browser, we use Puppeteer [87], a NodeJS library that provides a high-level API to control headless Chrome. During the runtime experiment, instead of playing the video, we record the metadata with each received chunk: its size, timestamp, and the quality level. We perform an offline analysis of the log files to calculate the throughput, latency, and accuracy.

Notice that the client-server relationship is reversed in HLS and AWStream. In HLS, the server hosts video source files; the client fetches videos and plays/computes. In AWStream, the client generates videos and pushes frames to the server; the server performs video analytics.

Why HLS/DASH is a poor match for video analytics? It is hard to achieve ultra low latency using HLS/DASH. (1) HLS/DASH is pull-based. The client keeps requesting for new chunks; (2) HLS/DASH uses chunking: shorter chunks enable a lower latency, but induce a larger number of requests, and the chunk has to be made ready before the client can start fetching. There are proposals to reduce the latency, such as server push [211], but they are still work in progress. For some applications, the accuracy can also be poor if these videos are limited to tuning resolution and encoding quality only, as demonstrated in our PD evaluation.

3.5.6 JetStream++

We modified the open source version of JetStream[7] in order to use our profile as its manual policy. Because JetStream doesn't support simultaneous degradation across multiple operators, we implemented a simple `VideoSource` operator that understands how to change image resolutions, frame rate, and video encoding quantization. At runtime, `VideoSource` queries congestion policy manager and adjusts three dimensions simultaneously. This operator is then exposed to the

[7] https://github.com/princeton-sns/jetstream/, commit bf0931b2d74d20fdf891669188feb84c96AF84.

Python-implemented control plane. We call this modified version JetStream++.[8]

JetStream's code base is modular and extensible: the modifications include 53 lines for the header file, 171 lines for implementation, 75 lines for unit test, and 49 lines of python as the application. While extending JetStream with our profile is not challenging, JetStream++ performs degradation in a single operator and loses the composability. We could modify JetStream to support degradation across multiple operators, but that would require substantial changes to JetStream. Using JetStream++ with our profile, the comparison is enough to illustrate the difference between AWStream's and JetStream's runtime.

3.6 Discussion

Reducing Developer Effort. While AWStream simplifies developing adaptive applications, there are still application-specific parts required for developers: wrapping appropriate `maybe` calls, providing training data, and implementing accuracy functions. Because AWStream's API is extensible, we plan to build libraries for common degradation operations and accuracy functions, similar to machine learning libraries.

Fault-tolerance and Recovery. AWStream tolerates bandwidth variation but not network partition or host failure. Although servers within data centers can handle faults in existing systems, such as Spark Streaming [218], it is difficult to make edge clients failure-oblivious. We leave failure detection and recovery as a future work.

Profile Modeling. AWStream currently performs an exhaustive search when profiling. While parallelism and sampling are effective, profiling complexity grows exponentially with the number of knobs. Inspired by recent success of using Bayesian Optimization [154, 184, 186] to model black-box functions, we can use multi-objective Bayesian Optimization (BO) [98] to search for *near-optimal* configurations. For applications in this chapter, exhaustive search is manageable. In the next chapter, for compute resource adaptation, the parameter space is prohibitively large and we explore the use of BO there.

Context detection. AWStream is currently limited to one profile: the offline profiling generates the default profile and the online profiling updates the single profile continuously. Real-world applications may produce data with a multi-modal distribution, where the model changes upon context changes, such as indoor versus outdoor. Therefore, one possible optimization to AWStream is to allow multiple profiles for one application, detect context changes, and use the profile that best matches the current data distribution. Switching contexts could reduce, or even eliminate, the overhead of online profiling.

Bandwidth Estimation and Prediction. Accurately estimating and predicting available bandwidth in wide area remains a challenge [103, 229]. AWStream uses network throughput and behaves cautiously to avoid building up queues: congestion is detected at both sender/receiver; data rate only

[8] https://github.com/awstream/jetstream-clone/pull/1

increases after probing. Recent research on adaptive video streaming explores model predictive control (MPC) [190, 215] and neural network [136]. We plan to explore these techniques next.

3.7 Chapter Summary

The emerging class of wide-area streaming analytics faces the challenge of scarce and variable bandwidth. Developers need to make a trade-off between data freshness and fidelity. While it is easy to build applications for either high accuracy or low latency, achieving both simultaneously is challenging. Previous approaches using manual policies end up with sub-optimal performance. Application-specific optimizations do not generalize to cover the variety and scales of potential applications.

This chapter presents AWStream, a stream processing system for a wide variety of applications by enabling developers to customize degradation operations with `maybe` operators. Our automatic profiling tool generates an accurate profile that characterizes the trade-off between bandwidth consumption and application accuracy. The profile allows the runtime to react with precision. Evaluations with three applications show that AWStream achieves sub-second latency with nominal accuracy drop. AWStream enables resilient execution of wide-area streaming analytics with minimal developer effort.

Chapter 4

Compute Resource Adaptation

In this chapter, we focus on swarm applications with computation-heavy tasks, such as machine learning (ML) inference. To create an engaging user experience, these applications need to offer bounded response times (e.g., 100 ms or sub-second). However, end devices are often not powerful enough to run an algorithm as is. While applications can offload computation to other resources such as the edge or the cloud, swarm platforms have a large discrepancy of computing capabilities. In addition, the edge/cloud faces the challenge of unpredictable network delays and service overload. To meet the response time requirement, we propose to adapt computation to each platform.

The ability to adapt comes from the availability of many options to for the same task: different algorithms or different parameters for the same algorithm. These options result in different processing times and application accuracy. If we can build a performance model that characterizes the impact of each option, the application can choose the right algorithm/parameter adaptively depending on the available compute power and meet the deadline.

Building this performance model for compute resource adaptation is challenging. Unlike the profiling discussed in Chapter 3, exhaustive profiling is not viable due to two issues: (i) we are facing algorithms with more parameters that form a much larger combinatorial space; (ii) because processing times vary across devices, we need to profile for each swarm device, even those that are not available at development time but will be used at deployment.

This chapter is step further in refining our profiling techniques. First, we address the issue with a large parameter space by employing Bayesian Optimization (BO). We demonstrate that BO can reduce the number of samples we need by 50$\times$ compared to an exhaustive approach. With the same search budget, BO finds better Pareto-optimal configurations than random search and coordinate search. Second, to profile for new devices, we propose to transfer an existing profile by updating the processing times of Pareto-optimal configurations. The profile transfer eliminates the need to get all training data, evaluate the algorithm for all data, and run BO engine.

4.1 Introduction

An increasing number of swarm applications rely on heavy computation tasks, notably machine learning (ML) applications. For example, Microsoft SeeingAI [140] is a talking camera for the blind and low vision community that describes nearby people, text, objects, etc. Wearable cognitive assistant [93] uses wearable devices, such as Google Glass, for deep cognitive assistance, e.g., offering hints for social interaction via real-time scene analysis.

While there is a staggering collection of research focusing on model *training* for these tasks, for swarm applications, they perform the *inference* step of machine learning: it uses a trained model to make a prediction given an input, such as visual [84, 93, 140], audio [10, 18, 139], and sensory information [122, 135]. Inference has received relatively little attention, especially on edge and end devices.

One important challenge in swarm applications is to bound the response time, which affects user experience significantly in these human-in-the-loop systems. Previous usability studies [149, 173] have measured the effect of different end-to-end response times:

- 100 ms gives the feeling of instantaneous response.
- 1 second keeps the user's flow of thought but users can sense the delay.
- A few seconds' delay creates an unpleasant user experience.

Achieving bounded response times (BRT) on swarm devices has remained challenging. Swarm devices are extremely heterogeneous and many are resource constrained (see Table 2.1). Heavy computations are often beyond the capabilities of cheap end devices. For example, state-of-the-art object detection algorithm easily takes several seconds on modern smartphones [48].

To compensate resource-constrained devices, prior work explores offloading [52, 58] to the cloud or the edge that hosts machine learning models and performs heavy computation. Despite their powerful computing capability and more available resources, e.g., special hardware like GPU and TPU [108], these machines suffer from increased network latency and service overload (Figure 2.2). As a result, they are often unable to meet response time goals consistently.

Another technique for low latency is to use redundant requests. By initiating multiple requests across a diverse set of resources and using the first result that completes, redundancy is effective to address the variability in network delays [88, 206] and slow servers (known as straggler mitigation in the cloud [14, 63]). However, existing works treat these resources equally and execute the same task on each platform, a poor match for the heterogeneous swarm space.

We make the observation that for a particular task, there are many options: different algorithms or tunable parameters for the same algorithm. These options result in different processing times and application accuracy. If we can adapt the computation by choosing an option that matches the capability of the device or satisfies the deadline requirement, we can achieve bounded response times.

The idea of adapting to machines is illustrated in Figure 4.1. It builds on top of redundancy but addresses, or rather leverages, the heterogeneity of swarm devices. On-device processing can use a

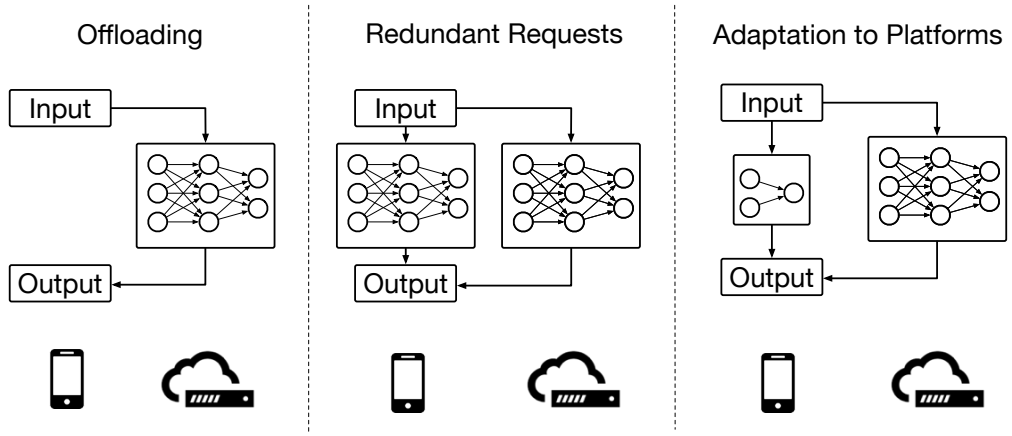


Figure 4.1: Illustration of offloading, redundant requests and compute adaptation.

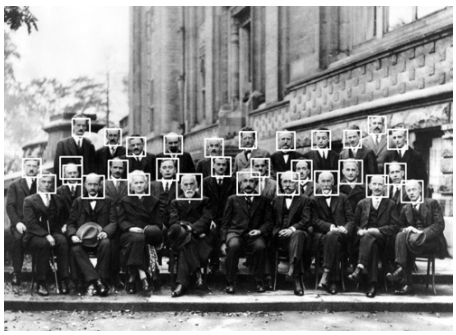
simple algorithm if the device is resource constrained. The device also sends redundant requests to other servers, e.g., the edge and/or the cloud. Each request contains a service-level objective (SLO) that describes the deadline and accuracy demand. Upon receiving the request, the server would admit or reject the request based on the network delays and its workload. Because of the redundancy, rejection is fine. This is different from traditional web serving where the server is the single point of source. At last, the device would use the first response that completes.

The key to enable adaptation lies in building an accurate performance model that characterized the processing times and accuracy trade-off for a particular task. We take a data-driven approach by measuring processing times and accuracy for a given set of training data. We call this process *profiling* and the resulting performance model is *the profile*.

Efficient profiling is challenging given the large space formed by the combination of algorithms, parameters, and machines. We need to profile each algorithm individually. For a single algorithm, it may have a huge parameter space. For example, cascading classifiers in computer vision tasks have configurable convolution window size, scale factor, and minimum neighbor for non maximum suppression. Exhaustive profiling in this space is prohibitively expensive (see Figure 4.5). Because processing times depend on the devices, the profile is different across devices. It is infeasible to profile all hardware platforms because of the staggering number of devices and their availability, or rather lack of, at development time.

In this chapter, we focus on improving profiling efficiency. We first describe our programming abstraction based on macro annotations. Developers only need to make small modifications to their existing code to express adaptation options for different algorithms and parameters. We then discuss techniques to build the performance model efficiently. Our profiling makes the following contributions,

1. To address the large parameter space, we use Bayesian Optimization (BO) for profiling. For Viola-Jones face detector, an exhaustive approach requires more than 4000 samples while BO learns the profile with around 80 samples—more than 50$\times$ fewer samples.



RPi Model B	Macbook Model A1502	Workstation Xeon E5-1620
4105 ms	544 ms	346 ms

Figure 4.2: (Left) Face detection with a photograph of the Fifth Solvay International Conference on Electrons and Photons. (Right) Processing times using Viola-Jones face detector on different machines,with default OpenCV parameters.

2. We make the observation that only processing times vary across devices. As a result, it is sufficient to only measure the processing times for configurations from the Pareto-optimal set. The profile transfer eliminates the need to get all training data, evaluate the algorithm for all data, and run BO engine.

4.2 Motivations

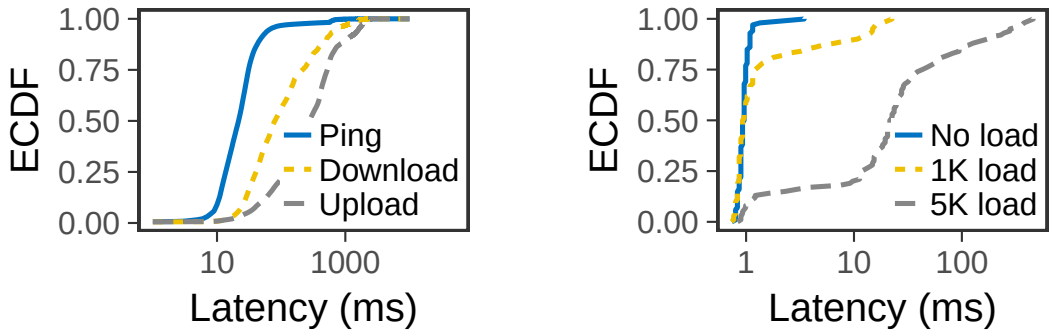
We focus on computation-heavy ML applications that are beyond the capabilities of end devices. In this chapter, we will first show the heterogeneity of swarm platforms for heavy computation and demonstrate that simple offloading does not provide consistent response times in the presence of network and workload variation. We then motivate computation adaptation by showing the trade-off between application accuracy and processing times in two cases: different algorithms and different parameters. At last, we summarize the challenges associated with building an accurate performance model.

4.2.1 Heterogeneous Environment

Our target application environment consists of machines with large range of computing resources. Earlier, §2.1.2 and Table 2.1 have discussed this dizzying array of machines ranging from powerful computing units to low-power microcontrollers. Low-power devices, such as mobile phones or IoT microcontrollers, are significantly limited in their processing capabilities. Performing ML inference easily takes seconds to complete. As shown in Figure 4.2, to detect faces in a photo of the Fifth Solvay International Conference on Electrons and Photons,[1] it takes more than 4 seconds on a Raspberry Pi (Model B).

One technique is to use the edge and/or the cloud for offloading. While they are substantially more powerful (7.5× to 11.9× in the face detection task), both the edge and the cloud suffer from

[1]The original image (3000×2171 pixels) is from Wikimedia and in the public domain.



(a) Broadband network latency.

(b) TensorFlow serving latency.

Figure 4.3: (Left) WAN network latency has variation and the latency deteriorates during downstream and upstream speed tests. (Right) Server processing latency has variation and the latency deteriorates during load increase.

variable latency, unstable connection, and service contention. These issues make it difficult to provide consistent response times, especially for tail performance. Figure 2.2 shows the characteristics of end devices, the edge, and the cloud. We then empirically validate the network variation and workload variation (Figure 4.3).

Network Variation. We use the raw data in January 2016 from FCC Measuring Broadband America Program [137] to validate the large variation in wide area network. Figure 4.3a shows the empirical cumulative distribution function (ECDF) of measured network latency. *Ping* time refers to the round trip time (RTT) of ICMP echo requests from measurement devices to a set of target test nodes. *Download* and *Upload* are latency measured when performing downstream or upstream speed tests. From this figure, we can see that the latency has 2-3 orders of magnitude difference. The situation is worse with active traffic. The median network delay increases from 22 ms to 80 ms under downstream load and 272 ms under upstream load.

Workload Variation. We measure end-to-end latency with TensorFlow serving [153], a state-of-the-art serving system for machine learning models. Specifically, we use MNIST [127] as a case study and study the serving performance with different level of background load. Figure 4.3b shows the ECDF with no load, 1K requests per second (RPS) and 5K RPS. From this figure, we can see that the serving latency has significantly increased when the load increases. With 1k load, 99.9 percentile latency increases from 3.5 ms to 22.5 ms. With 5K load, even the median latency increases to 21.5 ms: a 22.4× increase from 0.96 ms with no load.

4.2.2 Accuracy-Time Trade-off

Many tasks, especially ML inference, have multiple algorithms, or tunable parameters for the same algorithm. These options result in different application accuracy and processing times. As a result, we can speed up a computation task by sacrificing accuracy. In this way, many tasks become

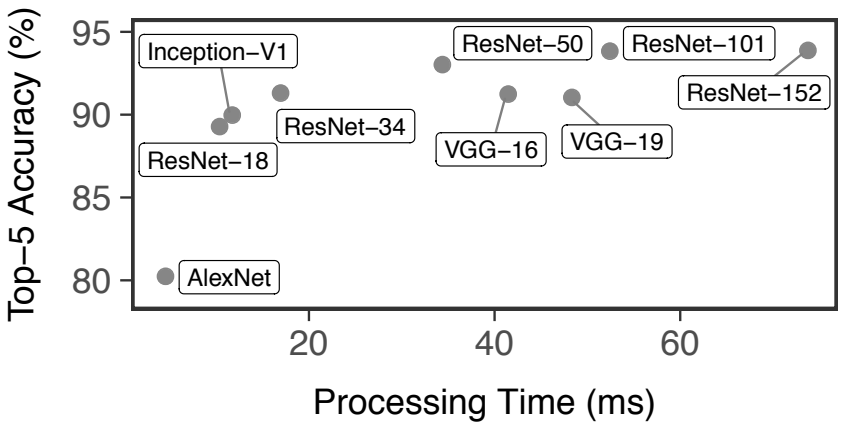


Figure 4.4: Benchmarks for popular CNN models demonstrate the trade-off between application accuracy and processing times. For data source and details, refer to cnn-benchmarks [107].

tractable on end devices. Servers can also tune their computation to accommodate network delays or address heavy load. Below we show two examples.

Convolutional Neural Networks (CNN). For the past few years, deep learning and neural networks have gained attention in performing complex machine learning tasks [83]. Convolutional neural networks (CNNs) are particularly powerful for computer vision tasks, such as object detection and image classification. There are many networks available: AlexNet [119], Inception [193], VGG [181], ResNet [96], etc. Huang et al. have observed the speed/accuracy trade-off and explored this trade-off in an exhaustive and fair manner [102]. Because this thesis is not an extensive study of neural networks, we only summarize one benchmark result in Figure 4.4. The processing times vary from 4.3 ms to 73.5 ms and the accuracy[2] varies from 80.2% to 93.8%.

Viola-Jones Face Detector. This is an example of how tunable parameters affect processing times and application accuracy. Viola-Jones (VJ) face detector [203] detects face by moving a window over the image. To detect faces at different sizes, it runs over an image pyramid with a configurable scale. After the detection, there are typically multiple neighboring windows with high scores close to the correct location of objects. A post-processing step uses non-maximum suppression (NMS) to group detection into a single bounding box. OpenCV [37] provides `CascadeClassifier` with the following parameters,

- `min_size`: minimum detectable object size.
- `scale`: how much image size is reduced at each image scale.
- `min_neighbors`: how many neighbors each candidate rectangle should have to retain it.

Figure 4.5 shows how these parameters affect processing times and accuracy. We measured the performance of 4,000 different parameter configurations[3] using FDDB dataset [105]. In the figure,

[2] Top-5 accuracy here means that the correct label is one of the top five predictions made by the network.

[3] `min_size` ranges from 0 to 190, with a step size of 10; `scale` ranges from 1.01 to 1.46, with a step size of

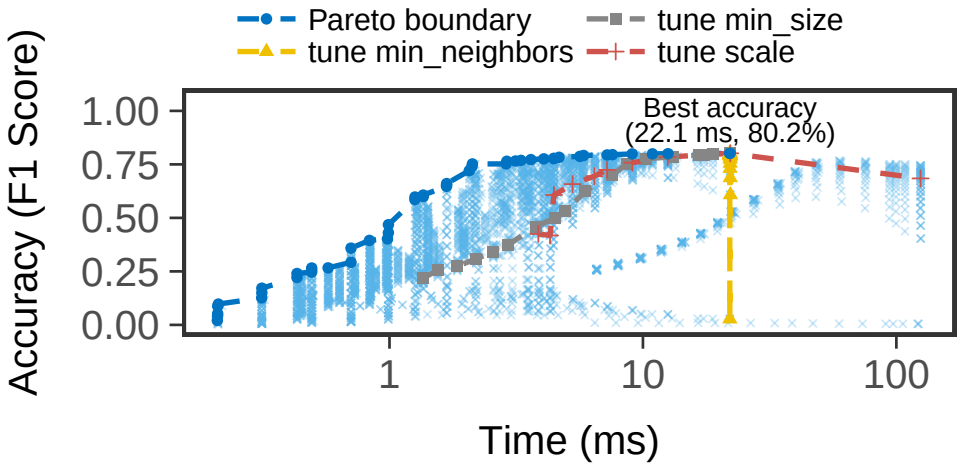


Figure 4.5: Benchmarks for Viola-Jones face detector with different parameters. These parameters form a large space that make performance modeling challenging.

each cross dot represents the processing time and application accuracy for one configuration. As we can see, there is a trade-off space: the processing times vary across almost 3 orders of magnitude; the accuracy can be as low as 0% and as high as 80.2%.

4.2.3 Performance Modeling and Challenges

We need to build the performance model to exploit the trade-off between accuracy and processing times. Because each algorithm is fundamentally different from another one, we must exhaustively evaluate all available algorithms. For the same algorithm with different parameters, exhaustive search can be prohibitively expensive as the parameters form a combinatorial space. Figure 4.5 shows an evaluation of 4,000 parameter configurations for Viola-Jones, which only has three parameters. A more complex algorithm can have a staggering number of tunable parameters. For example, HOG [59] in OpenCV has the following parameters: `win_size`, `win_stride`, `block_size`, `block_stride`, `cell_size`, `nbins`, `win_sigma`, `nlevels`, `hit_threshold`, `padding`, `group_threshold`, `use_meanshift_grouping`, etc.

A comprehensive performance modeling is actually not necessary. Instead, we only need configurations that provide the best accuracy for a given processing time constrain, i.e., the Pareto-optimal points. Figure 4.5 highlights the Pareto boundary in blue. We defer a formal definition of the Pareto-optimal set in §4.3.2.

In practice, we can reduce the number of samples further. The Pareto-optimal set only needs to be accurate enough to distinguish near-optimal configurations from the rest. Previous work has used two solutions: (i) *Random search.* This simple approach samples the parameter space randomly

0.05; `min_neighbors` ranges from 0 to 19, with a step size of 1.

and search/profile with a budget. It is obvious that as the budget increases, the coverage improves and the Pareto-optimal set is closer to the correct set. (ii) *Coordinate search.* This greedy approach starts with a random configuration, moves along one parameter dimension if it improves, and stops when it reaches a local optimum. After reaching the local optimum, if we still have search budget, this approach repeats by picking another random configuration.

Both random search and coordinate search have their problems. Random search needs a lot of samples to approximate the real Pareto boundary. Coordinate search may stuck in some local region and exhaust the search budget. As a result, with a limit budget, both approaches could return sub-optimal points in the trade-off space.

In addition, our discussion on performance modeling by far has not taken into account of the heterogeneous capabilities of devices. What we have shown in Figure 4.5 is measured on a workstation with Xeon E5-1620 CPU; the performance model will be different on another device, e.g., a Raspberry Pi.

4.3 Learning the Pareto-optimal Set Efficiently

In this section, we discuss techniques that learns the Pareto-optimal set efficiently. We first describe our programming abstraction and profiling formalism. We then focus on addressing two profiling challenges: the large parameter space and device heterogeneity. (i) We propose to use Bayesian Optimization (BO) to address large parameter space with multiple parameters. And we show that it outperforms previous approaches (random search and coordinate search). (ii) We propose to use profile transfer to derive the profile for a new device without running expensive profiling from scratch.

4.3.1 Programming Abstraction for Compute Adaptation

Unlike §3.3.1, the algorithms and parameters do not come from a series of chained operators. Therefore, we need new programming abstractions. However, the design goal is similar: to free developers from specifying exact rules and allow specifications with options.

We use `trait`[4] to abstract each algorithm, as below:

```
trait Algorithm {
    type I;
    type O;
    type P;
    fn execute(&mut self, i: &Self::I, p: &Self::P) -> Self::O;
}
```

Each algorithm is specified by its input data type `I`, output data type `O`, a parameter type `P`, and its behavior function `execute`. For any data type `T` that implements `Algorithm` trait, the

[4] `trait` in Rust is similar to `interface` in other languages.

```
#[derive(Adapt)]
struct VJParameters {
   #[adapt(min = "0", max = "20", step = "1")]
   min_neighbors: i64,

   #[adapt(min = "0", max = "190", step = "10")]
   min_size: i64,

   #[adapt(min = "1.01", max = "1.5", step = "0.01")]
   scale_factor: f64,
}

struct VJ {
  // ... omitted ...
}

impl Algorithm for VJ {
   type I = Image;
   type O = Detection;
   type P = VJParameters;

   fn execute(&mut self, image: &Image, opt: &VJParameters) -> Detection {
      // ... omitted ...
   }
}
```

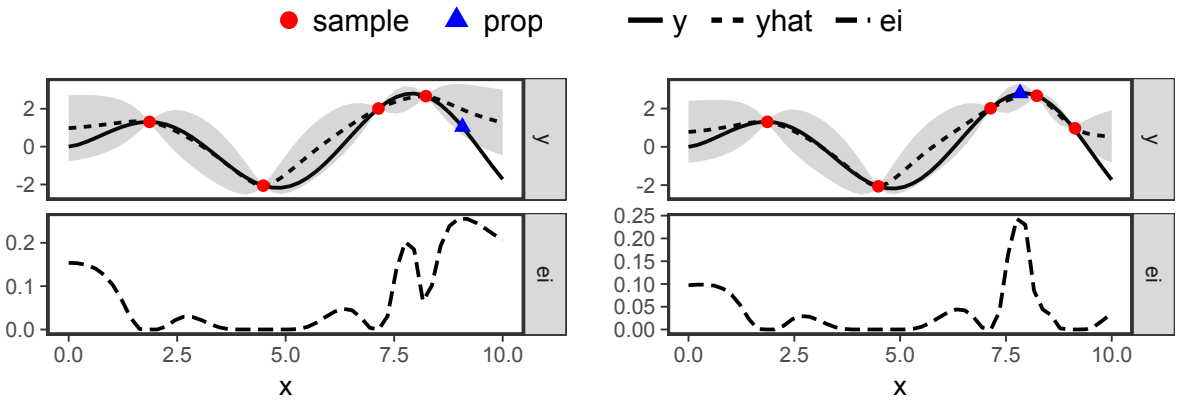
Figure 4.6: Viola-Jones face detector parameters `VJParameters` annotated with adaptation. `VJ` implements the `Algorithm` trait.

function `execute` can access the internal data structure of `T` in a mutable way: `&mut self`. Type `I`, `O`, and `P` are abstract types and the implementation must specify the concrete types. We show an implementation of Viola-Jones in Figure 4.6 where the input type is an image, the output type is a detection, and the parameter is `VJParameters`. When composing algorithms, their input data type and output data type must match. This type constrain is used to validate that two algorithms are performing the same task. Their parameters can be different.

For tunable parameters, we need to be able to specify at least three parts: `min`, `max`, and `step`. We extend an algorithm's parameter by augmenting it with macros annotations. In Rust, this can be implemented with procedural macros as a syntax extension. Figure 4.6 shows these annotations in blue: we use `#[derive(Adapt)]` to mark that this `struct` is a tunable parameter and use `#[adapt(...)]` to mark its field.

4.3.2 Profiling Formalism

The goal of profiling is to build a profile that characterizes the trade-off between application accuracy and processing times such that for a running application, it can choose the right algorithm/parameter



(a) First Iteration. **(b)** Second Iteration.

Figure 4.7: Bayesian Optimization illustrated.

to meet the deadline. As discussed in §4.2.3, we are only interested in the Pareto-optimal set.

We start with defining the Pareto-optimal set for a specific algorithm with varying parameters. For an algorithm with n parameters, e.g., n is 3 for Viola-Jones, we call each parameter a knob and refer it as k_i. The combination of all knobs forms a parameter *configuration* $c = [k_1, k_2, ...k_n]$. The set of all possible configurations $\mathbb{C}$ is the space that the profiling explores for this algorithm. For each configuration c, we are interested in two mappings: a mapping from c to its processing times $T(c)$ and its accuracy measure $A(c)$. The profiling looks for Pareto-optimal configurations; that is, for any configuration c in the Pareto-optimal set $\mathbb{P}$, there is no alternative configuration c' that requires a smaller processing time and offers a higher accuracy. Formally, $\mathbb{P}$ is defined as follows:

$$\mathbb{P} = \{c \in \mathbb{C} : \{c' \in \mathbb{C} : T(c') < T(c), A(c') > A(c)\} = \varnothing\} \tag{4.1}$$

Once we have the Pareto-optimal set for one algorithm, we can compose multiple algorithms and merge their Pareto-optimal set by finding the global Pareto-optimal configurations. Both processing times and application accuracy are simply real numbers and can be compared, although each algorithm's parameter may be completely different.

4.3.3 Profiling Parameters with Bayesian Optimization

Bayesian Optimization (BO) [184] is a technique that approximates black-box functions with proxy functions, iteratively proposes new sample point to evaluate in the large parameter space, and finds the global maximum (or minimum). It is effective for black-box functions when evaluating each sample is expensive and noisy, such as tuning parameters and model hyperparameters in ML tasks [184]. A comprehensive review of BO is beyond the scope of this thesis. We briefly describe what BO brings using an example without going into the gory details.

Figure 4.7, produced with the mlrMBO package [32], illustrates the process of finding the maximum point for 1-dimension Alpine N. 2 function [54]. The solid line labeled y is the target

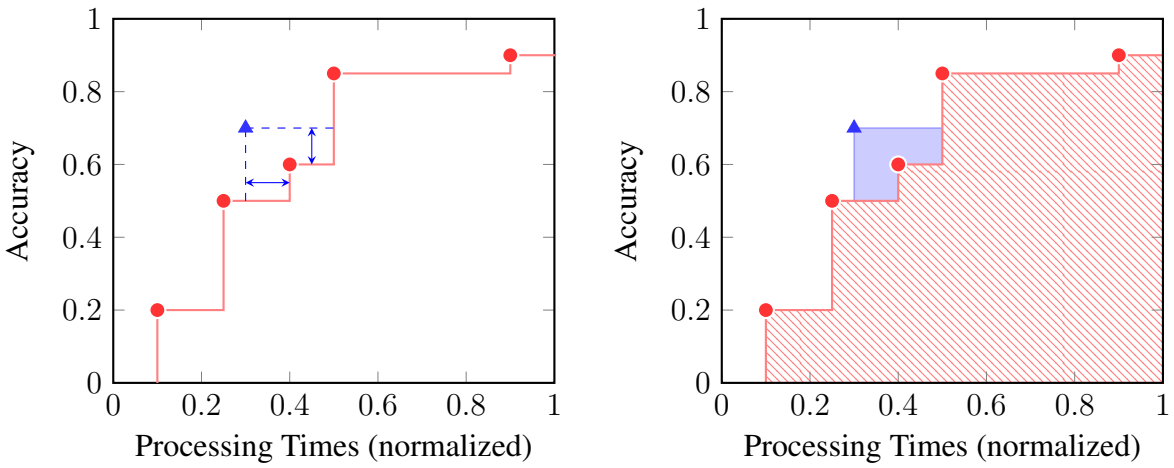


Figure 4.8: To find the Pareto-optimal set that maximizes accuracy while minimizing processing times, the acquisition criterion can suggest new samples (blue triangles) to the current Pareto-optimal set (red points) by maximizing additive epsilon (left, blue arrows) or hypervolume (right, blue shade area). This illustration is adapted from GPareto vignette [31].

black-box function. The red dots represent samples that have been evaluated. BO reconstructs a function that best fits these observed samples: the dashed line $\hat{y}$ (yhat) along with the confidence region (in gray shade). BO then uses an acquisition criteria to determine how to sample the next point. In this example, it uses expected improvement (EI, in long-dashed line) that balances high objective (exploitation) and high uncertainty (exploration) [178]. The next proposed sample is the point that maximizes EI (blue triangles). Running this process iteratively, BO is able to solve the maximization problem of a black-box function.

BO recently has been used beyond ML scope. CherryPick [154] uses BO to find the near-optimal cloud configurations for big data analytics, i.e., minimizing the cost of execution. Google uses BO to optimize chocolate chip cookies recipe [186], i.e., maximizes the tastiness.

Our problem is more complex as we are solving multi-objective optimizations (MOO). Specifically, it has two objectives: application accuracy and processing times. It is trivial to maximize application accuracy or minimize processing times alone. Previous approaches often transform multiple objectives into a single objective using scalarization techniques: using weighted sum to combine different objectives. These approaches are expected to be sub-optimal [116]. Recent research progress has made the transformation unnecessary. We can directly consider the Pareto-optimal set and evaluate the improvements with different acquisition criteria: (i) additive epsilon [31], (ii) hypervolume [31]; (iii) the entropy of the posterior distribution over the Pareto-optimal set [98]. We illustrate the first two criteria in Figure 4.8. We use PESMO [98] because of its availability and ease of use. It is available within the Spearmint package [183]. PESMO also has a low computational cost: it grows linearly with respective to the total number of objectives K.

We then show that BO can effectively explore the large design space and come up with a

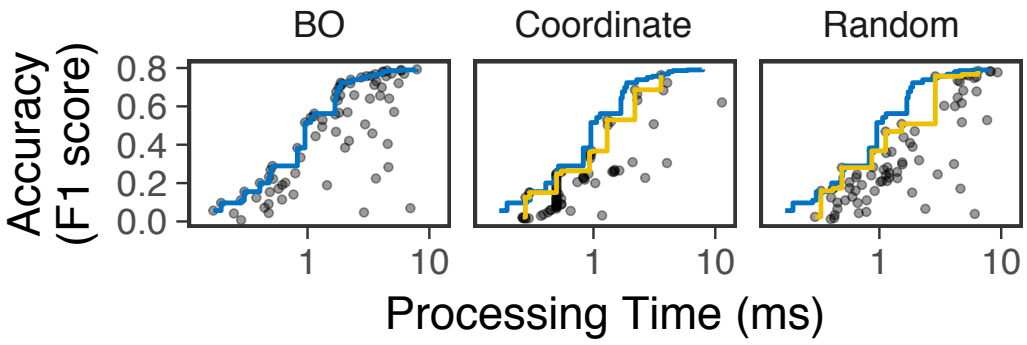


Figure 4.9: Using Viola-Jones as an example, BO evaluates 50 configurations and recommends 29 configurations as the Pareto-optimal boundary (the blue line in all subfigures). Greedy and Random find sub-optimal Pareto configurations with a budget of 80 evaluations (the yellow line in each figure).

better Pareto-optimal set. We compare BO with two baseline approaches: (i) random search; (ii) coordinate search.

For coordinate search, because it is a greedy approach, the original two objectives is converted into one single objective: $X(c) = A(c) - \beta T(c)$. We augment the basic version of the coordinate search with a few techniques: (i) to avoid starting with an expensive configuration and exploring its neighbors, we pick m random configurations and start from the one with the highest X. VideoStorm [224] suggests that even $m = 3$ can successfully avoid starting in an expensive part of the search space. (ii) because β affects the relative impact of T and A, we change its value once we have found a global maximum and still have search budget.

Figure 4.9 shows the evaluation results using Viola-Jones face detector and FDDB dataset. With similar search budgets, the Pareto-optimal set BO finds is better than the baseline approaches. We can see that coordinate search is relatively dense in some areas as it moves slowly from one point to its neighbors. Random search has a better coverage of the space, but to find a better Pareto front, it needs more samples.

4.3.4 Profile Transfer Across Devices

The profile we learned is specific to the devices. While it seems that we need to profile for every device, the profiling problem can be simplified. The insight lies in the properties of our objectives. $A(c)$ doesn't change across devices. We assume $T(c)$ is monotonic across devices, that is, a slower device remains slower for different configurations. Mathematically, if T_1 and T_2 are the mappings from configurations to processing times on two devices, the monotonic property means that if $T_1(c') < T_1(c)$, then $T_2(c') < T_2(c)$.

As a result, it is easy to see that the same configuration c remains in the Pareto-optimal set $\mathbb{P}$ regardless of what devices the algorithm runs on. So the whole Pareto-optimal set can be transferred. Visually, the profile on the second device is a stretched or shrunk version from a reference device

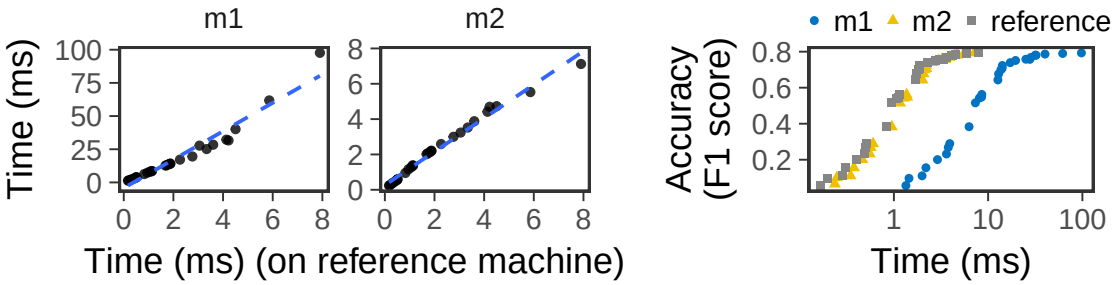


Figure 4.10: (Left) Empirically, processing times follows a linear approximation. (Right) Stretched/compressed profile. See paper for details.

along the dimension of processing times.

This simplifies our performance modeling on a new device. We only need to measure processing times of the Pareto-optimal set. And measuring processing times is significantly easier than measuring accuracy. We do not need to gather all the training data, run the algorithm, compare it with the ground-truth, or calculate the accuracy. The transfer can be further simplified if we use a linear transformation to approximate the relationship between processing times on two platforms. In this way, we only need to measure processing times from two configurations and extrapolate the rest.

We empirically validate profile transfer on three devices with Viola-Jones face detector (Figure 4.10). We use the profile learned on a reference machine and evaluate the processing times on two target machines. The left figure shows that a near-linear relationship between the process times on two machines. The right figure shows the stretched/shrunk effect of the profile. In this figure, we use the average processing times by running all the data. In practice, a new machine only needs to evaluate a few samples.

4.4 Discussion

More case studies. This chapter demonstrates the effectiveness of BO (Figure 4.9) and profile transfer (Figure 4.10). Our evaluation uses Viola-Jones face detector, a popular computer vision algorithm, as the primary case study. We have also applied our profiling to HOG [59] and the results are similar (hence omitted from this thesis). In the future, we expect to conduct more case studies with diverse algorithms.

Server scheduling. Our current use of the profile is simple: we find the algorithm/parameter that maximizes application accuracy given a processing time deadline. We envision that in a model serving system that handles a large number of requests, the profile can help with scheduling, e.g., the server tune processing times for some requests in order to maximize throughput.

4.5 Chapter Summary

Swarm applications that rely on computation-heavy tasks face the challenge of heterogeneous compute capabilities and variable network/service load. Without adaptation, the computation tasks would miss the deadline, which hurts user experience. In order to provide bounded response times, we propose to adapt computation based on a performance model that characterizes the trade-off between application accuracy and processing times.

Performance modeling for computation adaptation is challenging due to the large parameter space and heterogeneous device capabilities. This chapter focuses on improving the profiling efficiency with two techniques:

- We use Bayesian Optimization (BO) to address the large parameter space. BO models black-box function and iteratively proposes new sample points that are more likely to be in the Pareto-optimal set. Using Viola-Jones face detector as a case study, we demonstrate that BO reduces the number of samples needed by more than 50×, compared to exhaustive search. Compared with other proposed approaches (random search and coordinate search), BO finds better Pareto-optimal configurations with the same search budget.
- We use profile transfer to address heterogeneous device capabilities. New devices update an existing profile by only measuring the processing times of Pareto-optimal configurations. The profile transfer eliminates the need to get all training data, evaluate the algorithm for all data, and run BO engine. We have empirically validated profile transfer with VJ face detector.

The profile can guide an application to select an operating point that meets a response deadline. An advanced use of the profile is to guide a model serving system to make scheduling decisions. For example, the server can degrade the accuracy for a subset of requests to improve its overall throughput. In this thesis, we have focused on profiling, and the use of the profile at runtime is left as a future work.

Chapter 5

Related Work

In this thesis, we propose to use systematic adaptation and quantitative profiling to develop resilient swarm applications. We have summarized the trend of the emerging swarm in Chapter 2. In this chapter, we focus on research projects and industrial effort related to our techniques. We group related work into four categories: approximation and adaptation, taming constrained resources, efficient profiling, and other software systems.

5.1 Approximation and Adaptation

The idea of using approximation to trade application performance (such as accuracy) for system performance (such as response times) has been explored in a number of contexts. Hellerstein et al. [97] describes an online aggregation interface that provides approximate answers which are constantly refined as during execution. BlinkDB [6] enables interactive queries over massive data by allowing users to trade-off query accuracy. MEANTIME [77] uses approximate computing to achieve both hard real-time guarantees and energy efficiency. In these applications, partial or approximate results are more useful than late results. In addition to response times, some systems optimize for energy consumption. For example, EnerJ [168] uses programming language support to isolate parts of the program that must be precise from those that can be approximated to trade accuracy for energy. In this thesis, our research is along the same line: many swarm applications can reduce data fidelity or algorithm complexity, trading application accuracy for low latency.

For our work on network resource adaptation, the closest system to AWStream is JetStream [160]. JetStream is the first to use degradation to reduce latency for wide-area streaming analytics. However, JetStream relies on manual policies that are sub-optimal. AWStream instead proposes a novel API design to specify degradation and supports automatic profiling to search for Pareto-optimal configurations. AWStream also makes the following additional contributions: (1) online profiling to address model drift; (2) an improved runtime system achieving sub-second latency; (3) support for different resource allocation policies for multiple applications.

Adaptive video streaming is another domain in which network resource adaptation has been

extensively studied. Multimedia streaming protocols (e.g., RTP [176]) aim to be fast instead of reliable. While they can achieve low latency, their accuracy can be poor under congestion. Recent work has moved towards HTTP-based protocols and focused on designing adaptation strategy to improve QoE, as in research [136, 190, 215] and industry (HLS [156], DASH [138, 185]). These adaptation strategies are often pull-based: client keeps checking the index file for changes. And clients have to wait for the next chunk (typically 2-10 seconds). In addition, as we have shown in §3.5.3, their adaptation is a poor match for analytics that rely on image details (e.g., 6% accuracy for PD). In contrast, we propose to *learn* the adaptation strategy for each application (also not limited to video analytics).

The use of adaptation is not to provide hard Quality of Service (QoS) guarantees. In AWStream, we maximize application accuracy and minimize bandwidth requirement: a multi-dimensional optimization. And when the available bandwidth meets application requirement, we achieve low latency streaming. In addition, AWStream adopts an end-host application-layer approach ready today for WAN, while a large portion of network QoS work [78, 179, 180] in the 1990s focuses on network-layer solutions that are not widely deployable. For other application-layer approaches [202], we can apply our systematic adaptation approach by incorporating their techniques, such as encoding the number of layers as a knob to realize the layered approach in Rejaie et al. [164].

5.2 Taming Constrained Resources

In this thesis, we have addressed different types of constrained resources: (i) network resources, i.e., scarce and variable WAN bandwidth; (ii) compute resources, i.e., heterogeneous swarm devices. We also have studied the impact of variability, both network delays and service overload, in heavy-computation tasks. We discuss related work that addresses constrained resources.

Scarce and Variable WAN Bandwidth. Geo-distributed systems need to cope with high latency and limited bandwidth in the wide area. While some systems assume the network can prioritize a small amount of critical data under congestion [51], most systems reduce data sizes in the first place to avoid congestion (e.g., LBFS [146]). Over the past few years, we have seen a plethora of geo-distributed analytical frameworks that incorporate heterogeneous wide-area bandwidth into application execution to minimize data movement, as follows.

- Pixida [115] minimizes data movement across inter-DC links by solving a traffic minimization problem in data analytics.
- Iridium [158] optimizes data and task placement jointly to achieve an overall low latency goal.
- Geode [205] develops input data movement and join algorithm selection strategies to minimize WAN bandwidth usage.
- WANalytics [207] pushes computation to edge data centers, automatically optimizing workflow execution plans and replicating data only necessary.

- Clarinet [204] incorporates bandwidth information into its query optimizer to reduce data transfer.

While effective in speeding up analytics, these systems focus on batch tasks such as MapReduce jobs or SQL queries. Such tasks are executed infrequently and without real time constrain. In contrast, many swarm applications process streams of sensor data continuously in real time.

Offloading to Augment Mobile Devices. To improve application performance and extend battery life for mobile devices, research has explored offloading heavy-computation tasks to remote servers. Satyanarayanan used the word "cyber foraging" to refer to dynamically augmenting the computing resources of a wireless mobile computer by exploiting wired hardware infrastructure [170, 171].

Offloading receives much attention, especially after mobile computing took off since 2010. MAUI [58] supports fine-grained code offloading by exploiting the managed code environment. Developers annotate candidate methods to offload as `Remoteable`. MAUI then uses an optimization engine to decide which method to offload dynamically. CloneCloud [52] is a step further that partitions applications without developer annotation. In both MAUI and CloneCloud, the code either runs on a mobile device or on a server. Tango [88] explores the use of replication: the application executes on both the mobile device and the server. It allows either mobile devices or the server to lead the execution and uses flip-flop replication to allow leadership to float between replicas.

Redundancy to Tackle Latency Variability. The idea of initiating redundant operations across diverse resources and using the first result which completes has been explored in many contexts. For mobile computing, we have mentioned Tango [88] that replicates execution on both mobile devices and the server. Cloud computing often has straggler tasks that delay the job completion. Instead of waiting and trying to predict stragglers, Dolly [14] launches multiple clones of every task of a job and uses the result that finishes first. For cloud-based web services at Google, Dean et al. [63] describe similar redundancy techniques that address the tail performance at scale, such as hedged requests and tied requests. In wide-area network, Vulimiri et al. [206] show that redundancy reduces latency for a number of services, such as DNS queries, database queries, and packet forwarding.

For offloading, mobile devices and servers run the same algorithm (with the same configurations): either by partition or replication. For redundancy, an array of worker machines also run the same task. This fails to accommodate the heterogeneous swarm environment. The proposed adaptation in this thesis explicitly models performance across different devices. The learned profile can be used to assist offloading and replication.

5.3 Efficient Profiling

Prior work also use profiling to guide adaptation. For SQL queries, BlinkDB creates error-latency profiles to guide sample selection to meet the query's response time or accuracy requirements. For large scale video processing, VideoStorm [224] considers resource-quality profile and lag tolerance, and trades off between them.

For an efficient profiling, VideoStorm uses coordinate search—or greedy search in their terms. This thesis makes two major contributions: (1) our proposed BO outperforms coordinate search approach used in VideoStorm; (2) we handle heterogeneous devices while VideoStorm considered homogeneous machines in clusters.

BO, a technique for global optimization of black-box functions, has been applied to solve a wide range of problems. Snoek et al. [184] demonstrates its effectiveness in tuning parameters and modeling hyperparameters for ML tasks. CherryPick [154] uses BO to minimize cloud usage cost by searching for the best cloud configurations: the number of VMs, the number of cores, CPU speed per core, average RAM per core, disk speed, etc. FLASH [227] proposes a two-layer Bayesian optimization framework to tune hyperparameters in data analytic pipelines. BOAT [60] extends BO with by leveraging contextual information from developers and demonstrates the use in garbage collection and neural network. Google uses BO to optimize the recipe for chocolate chip cookies [186]. These use cases are single-objective optimization: minimizing time, cost or maximizing utility, tastiness. In this thesis, we face a multi-dimensional optimization between application accuracy and processing times.

5.4 Related Systems

AWStream has large debt to existing stream processing systems both for programming abstractions and handling back pressure. Early streaming databases [1, 46] have explored the use of dataflow models with specialized operators for stream processing. Recent research projects and open-source systems [7, 42, 120, 201, 218] primarily focus on fault-tolerance in the context of a single cluster. When facing back pressure, Storm [201], Spark Streaming [218] and Heron [120] throttle data ingestion; Apache Flink [42] uses edge-by-edge back-pressure techniques similar to TCP flow control; Faucet [125] leaves the flow control logic up to developers. In contrast, AWStream targets at the wide area and explicitly explores the trade-off between data fidelity and freshness.

Our compute resource adaptation work focuses on heavy-computation tasks, such as machine learning inference. Systems such as Clipper [57] and TensorFlow Serving [86] are state-of-the-art in serving ML models and performing inference. They only focus on system-level techniques such as batching and hardware acceleration to improve throughput. And our adaptation, especially the profile, can be used by these systems as application-aware improvements.

Chapter 6

Conclusion and Future Directions

This thesis claims that systematic adaptation and quantitative profiling are the key to a resilient swarm. This chapter summarizes our work, discusses future directions, and concludes this thesis.

6.1 Thesis Summary

This thesis began with an overview of the emerging swarm. As the swarm grows at a staggering rate, it faces challenges from the underlying communication and computation infrastructure: the scarce and variable WAN bandwidth and the heterogeneous compute environment. Existing approaches with manual policies or application-specific policies do not work for the swarm scale.

I propose to adapt swarm applications in a systematic and quantitative way. The proposed methodology consists of three parts: (i) a set of well-defined programming abstractions that allow developers to express what adaptation options are available; (ii) a data-driven profiling tool that automatically learns a Pareto-optimal profile; (iii) runtime adaptation that maximizes application performance while satisfying resource constrains. We apply our methodology to network resources and compute resources.

For network resources, swarm applications need to address the scarce and variable WAN bandwidth by exploring the trade-off between application accuracy and data size demand. In Chapter 3, we have presented our design, implementation, and evaluation of AWStream. Compared with non-adaptive applications, AWStream achieves a 40–100$\times$ reduction in latency relative to applications using TCP, and a 45–88% improvement in application accuracy relative to applications using UDP. Compared with manual policies using a state-of-the-art system JetStream [160], AWStream achieves a 15-20$\times$ reduction in latency and 1-5% improvement in accuracy simultaneously.

For compute resources, swarm applications need to address the heterogeneous compute environment by exploring the trade-off between application accuracy and processing times. In Chapter 4, we have focused on improving profiling efficiency. First, we use Bayesian Optimization to avoid an exhaustive exploration of the large parameter space. BO significantly reduces the number of samples needed, by more than 50$\times$ compared to an exhaustive profiling. With the same search budget,

BO finds better Pareto-optimal configurations than previous proposed search methods. Second, to profile for new devices, we propose to transfer an existing profile by updating the processing times of Pareto-optimal configurations. The profile transfer eliminates the need for get all training data, evaluate the algorithm for all data, and run a BO engine.

In summary, this thesis presents a systematic and quantitative approach for adaptation.

6.2 Future Directions

Before concluding this thesis, I discuss three directions that I find valuable as future work. Earlier in §3.6 and §4.4, we have discussed potential improvements that are near term and specific to our current design and implementation. This section focuses on future directions that are in a longer term and for a broader context.

More Diverse Applications and Generalization. The core challenge for the swarm comes from the scale of interconnected devices. As the number of devices continues to grow, we expect to see more diverse swarm applications. While we have used several applications in this thesis as case studies, we need more applications to evaluate how general the proposed APIs are, how effective the profiling techniques are, and how responsive the runtime system is. Some enhancements may be easily incorporated into our existing system. For example, because our APIs are extensible, we can build a library of reusable functions for accuracy measurements to reduce developer effort. Other enhancements may require a redesign of APIs, profiling tools, and/or the runtime.

Adaptation and Resource Allocation. In addition to adapting to resources, swarm applications should be able to dynamically *recruit* resources, such as sensors, actuators, data, and computing infrastructure. We have envisioned a SwarmOS that controls and allocates resources for different swarm applications. However, some swarm resources are beyond the control of individual entities (such as WAN bandwidth); some swarm resources are not as elastic as the cloud (such as end devices that need to be purchased and installed). In these cases, swarm applications need to adapt to such resources. One interesting direction is to use a combination of adaptation and resource allocation. In §3.3.4, we explored allocating available bandwidth among multiple applications and how different allocation schemes affect application behaviors. We believe this is an important future direction for the SwarmOS vision.

Adaptation and Mutation. The adaptation we have explored thus far is limited in what can be changed: applications only change the algorithms and parameters that a developer has specified as adaptation options. An extreme case for adaptation is self-modifying code, which is notoriously difficult to understand and manage. A slightly more flexible yet still disciplined approach is mutable accessors as discussed by Brooks et al. [38]. A mutable accessor is an abstract interface specification for candidate accessors. It reifies a concrete accessor downloaded from the Internet or retrieved through a discovery mechanism. Along the spectrum of adaptation, we expect future work to extend our profiling methodology to applications that change their structure and mutate their implementation.

6.3 Reflections and Concluding Remarks

In 2013, Gartner [141] positioned Internet of Things at the top of the hype cycle, i.e., peak of inflated expectations. In 2014–2015, the market was all about smart gadgets from crowdfunding projects [111] and startups [79]; large companies also began to build IoT platforms [12, 169]. Despite this blizzard of activities, IoT applications were not as disruptive as predicted. Many smart devices around this time provided questionable value, such as a smart bottle opener that messages your friends when you open a beer, a clothes peg that notifies you when your washing is dry, a smart egg tray that gives you a count of remaining eggs, etc [95]. Many products and companies then failed. A survey from Cisco estimated the failure rate to be around 75% [53]. In 2016–2017, public attention has transitioned to AI and blockchain. Interestingly, IoT still marches towards massive scale and actually benefits from the AI wave. With AI tackling speech recognition and object detection, it has become increasingly common to control smart devices through conversation with Alexa [10] and guard home security through person detection with security cameras [85]. Looking forward, there will be more research and industrial effort. The swarm vision will be slowly but steadily realized.

Developers face many challenges to get a swarm application running, even in normal conditions. It is often beyond their plans to address abnormal cases when the application needs to adapt. We have observed anecdotal cases when developers use manual and ad-hoc approaches for adaptation as quick fixes. The difficulties of developing swarm applications come from many factors. Due to missing platforms, developers have to dive into low-level, hardware-specific application design. Due to the fragmented and stove-piped ecosystem, developers spend non-trivial amount of time implementing adapters or wrappers for the interoperability of components. We expect to see more work on building swarm platforms [124, 144] and tackling interoperability [38]. And mature swarm platforms will lead to a proliferation of swarm applications. The adaptation approach described in this thesis will then allow resilient swarm applications that can work well even with insufficient resources.

This thesis proposes to adapt swarm applications with a systematic and quantitative approach. However, it must be noted that not all swarm applications necessarily need the proposed adaptation. Some swarm applications generate small amounts of data, e.g., millihertz sensors. Some swarm applications are delay tolerant, e.g., sensor data collection. In these cases, it is fine to develop applications as is. In addition, I would remark that our approach generalizes beyond swarm applications. While this thesis centers around swarm applications, an emerging category, we expect to apply this methodology in other contexts where applications need to cope with scarce and variable resources.

Bibliography

[1] Daniel J Abadi et al. “The Design of the Borealis Stream Processing Engine”. In: *CIDR*. Vol. 5. Asilomar, CA, 2005, pp. 277–289.

[2] Martín Abadi et al. “TensorFlow: A System for Large-scale Machine Learning”. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. GA: USENIX Association, 2016, pp. 265–283. ISBN: 978-1-931971-33-1. URL: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.

[3] Omid Abari et al. “Enabling High-Quality Untethered Virtual Reality”. In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 2017, pp. 531–544. ISBN: 978-1-931971-37-9. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/abari.

[4] Hervé Abdi. “The Kendall Rank Correlation Coefficient”. In: *Encyclopedia of Measurement and Statistics. Sage, Thousand Oaks, CA* (2007), pp. 508–510.

[5] Adobe. *High-quality, Network-efficient HTTP Streaming*. http://www.adobe.com/products/hds-dynamic-streaming.html. Accessed: 2017-09-20.

[6] Sameer Agarwal et al. “BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data”. In: *Proceedings of the 8th ACM European Conference on Computer Systems*. EuroSys ’13. Prague, Czech Republic: ACM, 2013, pp. 29–42. ISBN: 978-1-4503-1994-2. DOI: 10.1145/2465351.2465355.

[7] Tyler Akidau et al. “MillWheel: Fault-tolerant Stream Processing at Internet Scale”. In: *Proceedings of the VLDB Endowment* 6.11 (2013), pp. 1033–1044. URL: https://dl.acm.org/citation.cfm?id=2536229.

[8] Ian F Akyildiz et al. “Wireless Sensor Networks: A Survey”. In: *Computer networks* 38.4 (2002), pp. 393–422.

[9] Sara Alspaugh et al. “Analyzing Log Analysis: An Empirical Study of User Log Mining”. In: *Proceedings of the 28th USENIX Conference on Large Installation System Administration*. LISA’14. Seattle, WA: USENIX Association, 2014, pp. 53–68. ISBN: 978-1-931971-17-1. URL: http://dl.acm.org/citation.cfm?id=2717491.2717495.

[10] Amazon. *Alexa*. Accessed: 2017-08-05. 2017. URL: https://developer.amazon.com/alexa.

[11] Amazon. *Amazone EC2 Pricing*. https://aws.amazon.com/ec2/pricing/. Accessed: 2017-04-12. 2017.

[12] Amazon. *AWS IoT*. 2015. URL: https://aws.amazon.com/iot/.

[13] *Amazon S3 Durability*. http://aws.amazon.com/s3/faqs/#Data_Protection/. 2015.

[14] Ganesh Ananthanarayanan et al. "Effective Straggler Mitigation: Attack of the Clones". In: *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*. NSDI'13. Lombard, IL: USENIX Association, 2013, pp. 185–198. URL: http://dl.acm.org/citation.cfm?id=2482626.2482645.

[15] Michael P Andersen and David E Culler. "BTrDB: Optimizing Storage System Design for Timeseries Processing." In: *FAST*. 2016, pp. 39–52.

[16] Julia Angwin. *Has Privacy Become a Luxury Good?* http://www.nytimes.com/2014/03/04/opinion/has-privacy-become-a-luxury-good.html. 2014.

[17] Carles Anton-Haro and Mischa Dohler. *Machine-to-machine (M2M) Communications: Architecture, Performance and Applications*. Elsevier, 2014.

[18] Apple. *Siri*. Accessed: 2017-08-05. 2017. URL: https://www.apple.com/ios/siri/.

[19] *Arduino*. http://www.arduino.cc/.

[20] *ARM mbed*. https://mbed.org/.

[21] Michael Armbrust et al. "A View of Cloud Computing". In: *Communications of the ACM* 53.4 (2010), pp. 50–58.

[22] Roman Arutyunyan. *nginx-ts-module*. https://github.com/arut/nginx-ts-module. Accessed: 2017-09-20. 2017.

[23] Luigi Atzori, Antonio Iera, and Giacomo Morabito. "The Internet of Things: A Survey". In: *Computer networks* 54.15 (2010), pp. 2787–2805.

[24] *Automatic*. https://www.automatic.com/.

[25] Brian Babcock and Chris Olston. "Distributed Top-K Monitoring". In: *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*. SIGMOD'03. San Diego, California: ACM, 2003, pp. 28–39. ISBN: 1-58113-634-X. DOI: 10.1145/872757.872764.

[26] Naveen Balani and Rajeev Hathi. *Enterprise IoT: A Definitive Handbook*. CreateSpace Independent Publishing Platform, 2016.

[27] Jonathan Bar-Magen. “Fog Computing: Introduction to a New Cloud Evolution”. In: *Escrituras silenciadas: paisaje como historiografía*. Servicio de Publicaciones. 2013, pp. 111–126.

[28] Jeff Barr. *Amazon S3 – Two Trillion Objects, 1.1 Million Requests / Second*. https://aws.amazon.com/blogs/aws/amazon-s3-two-trillion-objects-11-million-requests-second/. 2013.

[29] Fabrice Bellard, M Niedermayer, et al. “FFmpeg”. In: (2012).

[30] Alysson Bessani et al. “DepSky: dependable and secure storage in a cloud-of-clouds”. In: *ACM Transactions on Storage (TOS)* 9.4 (2013), p. 12.

[31] Mickaël Binois and Victor Picheny. “GPareto: An R Package for Gaussian-Process Based Multi-Objective Optimization and Analysis”. In: (2018).

[32] Bernd Bischl et al. “mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions”. In: *arXiv preprint arXiv:1703.03373* (2017).

[33] Sanjit Biswas et al. “Large-scale Measurements of Wireless Network Behavior”. In: *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. SIGCOMM’15. London, United Kingdom: ACM, 2015, pp. 153–165. ISBN: 978-1-4503-3542-3. DOI: 10.1145/2785956.2787489.

[34] Robert Bogue. “Towards the Trillion Sensors Market”. In: *Sensor Review* 34.2 (2014), pp. 137–142.

[35] Flavio Bonomi et al. “Fog Computing and Its Role in the Internet of Things”. In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM. 2012, pp. 13–16.

[36] Joseph Bradley et al. “Internet of Everything (IoE)”. In: *Survey Report* (2013).

[37] G. Bradski. “The OpenCV Library”. In: *Doctor Dobbs Journal* (2000–2017). URL: http://opencv.org.

[38] Christopher Brooks et al. “A Component Architecture for the Internet of Things”. In: *Proceedings of the IEEE* (2018).

[39] AJ Brush et al. “Lab of things: a platform for conducting studies with connected devices in multiple homes”. In: *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*. ACM. 2013, pp. 35–38.

[40] Matt Calder et al. “Mapping the Expansion of Google’s Serving Infrastructure”. In: *Proceedings of the 2013 Conference on Internet Measurement Conference*. IMC’13. Barcelona, Spain: ACM, 2013, pp. 313–326. ISBN: 978-1-4503-1953-9. DOI: 10.1145/2504730.2504754.

[41] Pei Cao and Zhe Wang. “Efficient Top-K Query Calculation in Distributed Networks”. In: *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*. ACM. 2004, pp. 206–215.

[42] Paris Carbone et al. “Apache flink: Stream and batch processing in a single engine”. In: *Data Engineering* 38.4 (2015). URL: https://flink.apache.org/.

[43] Neal Cardwell et al. “BBR: Congestion-based Congestion Control”. In: *Communications of the ACM* 60.2 (2017), pp. 58–66. URL: http://dl.acm.org/citation.cfm?id=3042068.3009824.

[44] *Carriots*. https://www.carriots.com/.

[45] Geoffrey Challen et al. “The Mote is Dead. Long Live the Discarded Smartphone!” In: *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications*. ACM. 2014, p. 5.

[46] Sirish Chandrasekaran et al. “TelegraphCQ: Continuous Dataflow Processing”. In: *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*. SIGMOD’03. San Diego, California: ACM, 2003, pp. 668–668. ISBN: 1-58113-634-X. DOI: 10.1145/872757.872857.

[47] Kaifei Chen et al. “SnapLink: Fast and Accurate Vision-Based Appliance Control in Large Commercial Buildings”. In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1.4 (2018), p. 129.

[48] Tiffany Yu-Han Chen et al. “Glimpse: Continuous, Real-time Object Recognition on Mobile Devices”. In: *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. ACM. 2015, pp. 155–168.

[49] Zhuo Chen. “An Application Platform for Wearable Cognitive Assistance”. PhD thesis. Carnegie Mellon University, 2018.

[50] Yun Cheng et al. “AirCloud: A Cloud-based Air-Quality Monitoring System for Everyone”. In: *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*. ACM. 2014, pp. 251–265.

[51] Brian Cho and Marcos K Aguilera. “Surviving Congestion in Geo-Distributed Storage Systems”. In: *USENIX Annual Technical Conference (USENIX ATC 12)*. USENIX, 2012, pp. 439–451. URL: https://www.usenix.org/conference/atc12/technical-sessions/presentation/cho.

[52] Byung-Gon Chun et al. “Clonecloud: Elastic Execution Between Mobile Device and Cloud”. In: *Proceedings of the sixth conference on Computer systems*. ACM. 2011, pp. 301–314.

[53] Cisco. *The Journey to IoT Value*. https://www.slideshare.net/CiscoBusinessInsights/journey-to-iot-value-76163389. 2017.

[54] Maurice Clerc. “The Swarm and the Queen, Towards a Deterministic and Adaptive Particle Swarm Optimization”. In: *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*. Vol. 3. IEEE. 1999, pp. 1951–1957.

[55] Benjamin Coifman et al. “A Real-time Computer Vision System for Vehicle Tracking and Traffic Surveillance”. In: *Transportation Research Part C: Emerging Technologies* 6.4 (1998), pp. 271–288.

[56] FlexRay Consortium et al. “FlexRay Communications System-protocol Specification”. In: *Version* 2.1 (2005), pp. 198–207.

[57] Daniel Crankshaw et al. “Clipper: A Low-Latency Online Prediction Serving System”. In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 2017, pp. 613–627. ISBN: 978-1-931971-37-9. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/crankshaw.

[58] Eduardo Cuervo et al. “MAUI: Making Smartphones last Longer With Code Offload”. In: *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM. 2010, pp. 49–62.

[59] Navneet Dalal and Bill Triggs. “Histograms of Oriented Gradients for Human Detection”. In: *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*. CVPR '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 886–893. ISBN: 0-7695-2372-2. DOI: 10.1109/CVPR.2005.177.

[60] Valentin Dalibard, Michael Schaarschmidt, and Eiko Yoneki. “BOAT: Building Auto-Tuners with Structured Bayesian Optimization”. In: *Proceedings of the 26th International Conference on World Wide Web*. WWW'17. Perth, Australia: International World Wide Web Conferences Steering Committee, 2017, pp. 479–488. ISBN: 978-1-4503-4913-0. DOI: 10.1145/3038912.3052662.

[61] Stephen Dawson-Haggerty et al. “BOSS: Building Operating System Services”. In: *NSDI*. Vol. 13. 2013, pp. 443–458.

[62] Stephen Dawson-Haggerty et al. “sMAP: a simple measurement and actuation profile for physical information”. In: *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*. ACM. 2010, pp. 197–210.

[63] Jeffrey Dean and Luiz André Barroso. “The Tail at Scale”. In: *Communications of the ACM* 56.2 (2013), pp. 74–80.

[64] DERA. *EDGAR Log File Data Set*. https://www.sec.gov/data/edgar-log-file-data-set. Accessed: 2017-01-25. 2003–2016.

[65] video dev. *hls.js*. https://github.com/video-dev/hls.js. Accessed: 2017-09-20. 2017.

[66] Martin Devera. *HTB Home*. http://luxik.cdi.cz/~devik/qos/htb/. Accessed: 2017-04-08. 2001–2003.

[67] Martin Diaz, Gonzalo Juan, and Alberto Ryuga Oikawa Lucas. “Big Data on the Internet of Things”. In: *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. 2012, pp. 978–0.

[68] Piotr Dollar et al. “Pedestrian Detection: An Evaluation of the State of the Art”. In: *IEEE transactions on pattern analysis and machine intelligence* 34.4 (2012), pp. 743–761.

[69] Dale Dougherty. “The Maker Movement”. In: *Innovations: Technology, Governance, Globalization* 7.3 (2012), pp. 11–14.

[70] Alex Drozhzhin. *Internet of Crappy Things*. http://blog.kaspersky.com/internet-of-crappy-things/. 2015.

[71] Arjan Durresi and Raj Jain. *RTP, RTCP, and RTSP-Internet Protocols for Real-Time Multimedia Communication.* 2005.

[72] John C Eidson et al. “Distributed real-time software for cyber–physical systems”. In: *Proceedings of the IEEE* 100.1 (2012), pp. 45–59.

[73] Martin Eigner. “The Industrial Internet”. In: *The Internet of Things*. Springer, 2018, pp. 133–169.

[74] ESnet. *iPerf: The TCP/UDP bandwidth measurement tool*. http://software.es.net/iperf/. Accessed: 2017-03-07. 2014-2017.

[75] Dave Evans. “The Internet of Things: How the Next Evolution of the Internet is Changing Everything”. In: *CISCO white paper* 1 (2011).

[76] Mark Everingham et al. “The Pascal Visual Object Classes (VOC) Challenge”. In: *Int. J. Comput. Vision* 88.2 (June 2010), pp. 303–338. ISSN: 0920-5691. DOI: 10.1007/s11263-009-0275-4.

[77] Anne Farrell and Henry Hoffmann. “MEANTIME: Achieving Both Minimal Energy and Timeliness with Approximate Computing”. In: *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. Denver, CO: USENIX Association, 2016, pp. 421–435. ISBN: 978-1-931971-30-0. URL: https://www.usenix.org/conference/atc16/technical-sessions/presentation/farrell.

[78] Domenico Ferrari and Dinesh C Verma. “A Scheme for Real-time Channel Establishment in Wide-area Networks”. In: *IEEE journal on Selected Areas in communications* 8.3 (1990), pp. 368–379.

[79] *Fitbit*. http://www.fitbit.com/.

[80] AT&T Labs & AT&T Foundry. “AT&T Edge Cloud (AEC) - White Paper”. In: (2017).

[81] FTC. *Internet of Things, Privacy & Security in a Connected World*. http://www.ftc.gov/system/files/documents/reports/federal-trade-commission-staff-report-november-2013-workshop-entitled-internet-things-privacy/150127iotrpt.pdf. 2015.

[82] GE. *Industrial Internet Insights*. https://www.ge.com/digital/industrial-internet. Accessed: 2017-09-23. 2017.

[83] Ian Goodfellow et al. *Deep Learning*. Vol. 1. MIT press Cambridge, 2016.

[84] Google. *Google Lens*. Accessed: 2017-08-05. 2017. URL: https://en.wikipedia.org/wiki/Google_Lens.

[85] Google. *Nest Cam Indoor*. https://www.dropcam.com. Accessed: 2017-04-03. 2009-2017.

[86] Google. *TensorFlow Serving*. 2017. URL: https://www.tensorflow.org/serving/.

[87] Google Chrome. *Puppeteer*. https://github.com/GoogleChrome/puppeteer. Accessed: 2017-09-20. 2017.

[88] Mark S. Gordon et al. "Accelerating Mobile Applications Through Flip-Flop Replication". In: *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys'15. Florence, Italy: ACM, 2015, pp. 137–150. ISBN: 978-1-4503-3494-5. DOI: 10.1145/2742647.2742649.

[89] Sarthak Grover et al. "Peeking Behind the NAT: An Empirical Study of Home Networks". In: *Proceedings of the 2013 Conference on Internet Measurement Conference*. IMC'13. Barcelona, Spain: ACM, 2013, pp. 377–390. ISBN: 978-1-4503-1953-9. DOI: 10.1145/2504730.2504736.

[90] *GroveStreams*. https://www.grovestreams.com/.

[91] Sidhant Gupta, Eric Larson, and Shwetak Patel. *Household Energy Dataset*. http://ubicomplab.cs.washington.edu/projects/datasets. 2015.

[92] Trinabh Gupta, Rayman Preet Singh, and Amar Phanishayee Jaeyeon Jung Ratul Mahajan. "Bolt: Data management for connected homes". In: *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. 2014, pp. 243–256.

[93] Kiryong Ha et al. "Towards Wearable Cognitive Assistance". In: *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys'14. Bretton Woods, New Hampshire, USA: ACM, 2014, pp. 68–81. ISBN: 978-1-4503-2793-0. DOI: 10.1145/2594368.2594383.

[94] George William Hart. "Nonintrusive Appliance Load Monitoring". In: *Proceedings of the IEEE* 80.12 (1992), pp. 1870–1891.

[95] Björn Hartmann. *The societal costs of the IOT outweigh its benefits*. http://iot.stanford.edu/retreat16/sitp16-benefit-no.pdf. 2016.

[96] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[97] Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. "Online Aggregation". In: *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*. SIGMOD'97. Tucson, Arizona, USA: ACM, 1997, pp. 171–182. ISBN: 0-89791-911-4. DOI: 10.1145/253260.253291.

[98] Daniel Hernández-Lobato et al. "Predictive Entropy Search for Multi-Objective Bayesian Optimization". In: *International Conference on Machine Learning*. 2016, pp. 1492–1501.

[99] Timothy W Hnat et al. "The hitchhiker's guide to successful residential sensing deployments". In: *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*. ACM. 2011, pp. 232–245.

[100] Jin-Hyuk Hong, Ben Margines, and Anind K Dey. "A smartphone-based sensing platform to model aggressive driving behaviors". In: *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*. ACM. 2014, pp. 4047–4056.

[101] Kevin Hsieh et al. "Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds". In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association. USENIX Association, 2017. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/hsieh.

[102] Jonathan Huang et al. "Speed/accuracy Trade-offs for Modern Convolutional Object Detectors". In: *arXiv preprint arXiv:1611.10012* (2016).

[103] Te-Yuan Huang et al. "Confused, Timid, and Unstable: Picking a Video Streaming Rate is Hard". In: *Proceedings of the 2012 Internet Measurement Conference*. IMC '12. Boston, Massachusetts, USA: ACM, 2012, pp. 225–238. ISBN: 978-1-4503-1705-4. DOI: 10.1145/2398776.2398800.

[104] Bert Hubert. *Linux Advanced Routing & Traffic Control*. http://lartc.org/. Accessed: 2017-04-06. 2002.

[105] Vidit Jain and Erik Learned-Miller. *FDDB: A Benchmark for Face Detection in Unconstrained Settings*. Tech. rep. UM-CS-2010-009. University of Massachusetts, Amherst, 2010.

[106] *JanOS: Turn your phone into an IoT board*. http://janos.io/.

[107] Justin Johnson. *Benchmarks for popular CNN models*. 2016. URL: https://github.com/jcjohnson/cnn-benchmarks/.

[108] Norman P Jouppi et al. "In-datacenter Performance Analysis of a Tensor Processing Unit". In: *arXiv preprint arXiv:1704.04760* (2017).

[109] Joseph M Kahn, Randy H Katz, and Kristofer SJ Pister. "Next Century Challenges: Mobile Networking for "Smart Dust"". In: *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*. ACM. 1999, pp. 271–278.

[110] David Karger, Cliff Stein, and Joel Wein. *Algorithms and Theory of Computation Handbook*. Chapman & Hall/CRC, 2010. Chap. Scheduling Algorithms, pp. 20–20. ISBN: 978-1-58488-820-8. URL: http://dl.acm.org/citation.cfm?id=1882723.1882743.

[111] *KickStarter*. https://www.kickstarter.com/.

[112] Hokeun Kim. "Securing the Internet of Things via Locally Centralized, Globally Distributed Authentication and Authorization". PhD thesis. EECS Department, University of California, Berkeley, 2017. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-139.html.

[113] Sukun Kim et al. “Health Monitoring of Civil Infrastructures Using Wireless Sensor Networks”. In: *Proceedings of the 6th international conference on Information processing in sensor networks*. ACM. 2007, pp. 254–263.

[114] Joel W King. *Cisco IP Video Surveillance Design Guide*. https://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Video/IPVS/IPVS_DG/IPVS-DesignGuide.pdf. 2009.

[115] Konstantinos Kloudas et al. “Pixida: optimizing data parallel jobs in wide-area data analytics”. In: *Proceedings of the VLDB Endowment* 9.2 (2015), pp. 72–83. URL: https://dl.acm.org/citation.cfm?id=2850582.

[116] Joshua Knowles. “ParEGO: A Hybrid Algorithm with On-line Landscape Approximation for Expensive Multiobjective Optimization Problems”. In: *IEEE Transactions on Evolutionary Computation* 10.1 (2006), pp. 50–66.

[117] J Zico Kolter and Matthew J Johnson. “REDD: A public data set for energy disaggregation research”. In: *Workshop on Data Mining Applications in Sustainability (SIGKDD), San Diego, CA*. Vol. 25. Citeseer. 2011, pp. 59–62.

[118] Andrew Krioukov et al. “Building Application Stack (BAS)”. In: *Proceedings of the Fourth ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*. BuildSys ’12. Toronto, Ontario, Canada: ACM, 2012, pp. 72–79. ISBN: 978-1-4503-1170-0. DOI: 10.1145/2422531.2422546.

[119] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet Classification with Deep Convolutional Neural Networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[120] Sanjeev Kulkarni et al. “Twitter Heron: Stream Processing at Scale”. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. SIGMOD’15. Melbourne, Victoria, Australia: ACM, 2015, pp. 239–250. ISBN: 978-1-4503-2758-9. DOI: 10.1145/2723372.2742788.

[121] Frank Langfitt. *In China, Beware: A Camera May Be Watching You*. http://www.npr.org/2013/01/29/170469038/in-china-beware-a-camera-may-be-watching-you. Accessed: 2017-04-04. 2013.

[122] Gierad Laput, Yang Zhang, and Chris Harrison. “Synthetic Sensors: Towards General-Purpose Sensing”. In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM. 2017, pp. 3986–3999.

[123] Heiner Lasi et al. “Industry 4.0”. In: *Business & Information Systems Engineering* 6.4 (2014), pp. 239–242.

[124] E. Latronico et al. “A Vision of Swarmlets”. In: *Internet Computing, IEEE* PP.99 (2015), pp. 1–1. ISSN: 1089-7801. DOI: 10.1109/MIC.2015.17.

[125] Andrea Lattuada, Frank McSherry, and Zaheer Chothia. "Faucet: a User-Level, Modular Technique for Flow Control in Dataflow Engines". In: *Proceedings of the 3rd ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond*. ACM. 2016, p. 2.

[126] *Learning at Adafruit Industries, Unique & Fun DIY Electronics and Kits*. https://learn.adafruit.com.

[127] Yann LeCun. "The MNIST database of handwritten digits". In: *http://yann. lecun. com/exdb/mnist/* (1998).

[128] Edward A. Lee. "IoT and CPS: A Focus on Modeling". In: *U.S.-German Workshop on the Internet of Things (IoT) / Cyber-Physical Systems (CPS)*. 2016.

[129] Edward A Lee. "The past, present and future of cyber-physical systems: A focus on models". In: *Sensors* 15.3 (2015), pp. 4837–4869.

[130] Edward A Lee. "What Is Real Time Computing? A Personal View". In: *IEEE DESIGN AND TEST* 35.2 (2018), pp. 64–72.

[131] Edward A. Lee et al. *The TerraSwarm Research Center (TSRC) (A White Paper)*. Tech. rep. UCB/EECS-2012-207. EECS Department, University of California, Berkeley, 2012. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-207.html.

[132] Ang Li et al. "CloudCmp: Comparing Public Cloud Providers". In: *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*. IMC'10. Melbourne, Australia: ACM, 2010, pp. 1–14. ISBN: 978-1-4503-0483-2. DOI: 10.1145/1879141.1879143.

[133] David G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: *Int. J. Comput. Vision* 60.2 (Nov. 2004), pp. 91–110. ISSN: 0920-5691. DOI: 10.1023/B:VISI.0000029664.99615.94.

[134] Chaochao Lu and Xiaoou Tang. "Surpassing Human-level Face Verification Performance on LFW with Gaussian Face". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI'15. Austin, Texas: AAAI Press, 2015, pp. 3811–3819. ISBN: 0-262-51129-0. URL: https://dl.acm.org/citation.cfm?id=2888245.

[135] Hong Lu et al. "The Jigsaw Continuous Sensing Engine for Mobile Phone Applications". In: *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*. SenSys'10. Zürich, Switzerland: ACM, 2010, pp. 71–84. ISBN: 978-1-4503-0344-6. DOI: 10.1145/1869983.1869992.

[136] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. "Neural Adaptive Video Streaming with Pensieve". In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM. 2017, pp. 197–210.

[137] *Measuring Broadband America*. https://www.fcc.gov/general/measuring-broadband-america.

[138] MG Michalos, SP Kessanidis, and SL Nalmpantis. "Dynamic Adaptive Streaming over HTTP". In: *Journal of Engineering Science and Technology Review* 5.2 (2012), pp. 30–34.

[139] Microsoft. *Cortana | Your Intelligent Virtual & Personal Assistant*. Accessed: 2017-08-05. 2017. URL: https://www.microsoft.com/en-us/windows/cortana.

[140] Microsoft. *Seeing AI*. Accessed: 2017-08-02. 2017. URL: https://www.microsoft.com/en-us/seeing-ai/.

[141] Peter Middleton, Peter Kjeldsen, and Jim Tully. "Forecast: The Internet of Things, Worldwide, 2013". In: *Gartner Research* (2013).

[142] Anton Milan et al. "MOT16: A Benchmark for Multi-Object Tracking". In: *arXiv preprint arXiv:1603.00831* (2016). URL: https://motchallenge.net/.

[143] Robert B Miller. "Response Time in Man-computer Conversational Transactions". In: *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*. ACM. 1968, pp. 267–277.

[144] Nitesh Mor et al. "Toward a Global Data Infrastructure". In: *IEEE Internet Computing* 20.3 (2016), pp. 54–62.

[145] Matthew K Mukerjee et al. "Practical, Real-time Centralized Control for CDN-based Live Video Delivery". In: *ACM SIGCOMM Computer Communication Review* 45.4 (2015), pp. 311–324. URL: https://dl.acm.org/citation.cfm?id=2787475.

[146] Athicha Muthitacharoen, Benjie Chen, and David Mazières. "A Low-bandwidth Network File System". In: *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*. SOSP'01. Banff, Alberta, Canada: ACM, 2001, pp. 174–187. ISBN: 1-58113-389-8. DOI: 10.1145/502034.502052.

[147] Cisco Visual Networking. "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016-2021 White Paper". In: *Cisco White Paper* (2016).

[148] Cisco Visual Networking. "The Zettabyte Era: Trends and Analysis". In: *Cisco White Paper* (2013).

[149] Jakob Nielsen. *Usability Engineering*. Elsevier, 1994.

[150] Ashkan Nikravesh et al. "Mobile Network Performance from User Devices: A Longitudinal, Multidimensional Analysis". In: *Proceedings of the 15th International Conference on Passive and Active Measurement - Volume 8362*. PAM 2014. Los Angeles, CA, USA: Springer-Verlag New York, Inc., 2014, pp. 12–22. ISBN: 978-3-319-04917-5. DOI: 10.1007/978-3-319-04918-2_2.

[151] *Ninja Blocks*. https://ninjablocks.com/.

[152] Sangmin Oh et al. "A Large-scale Benchmark Dataset for Event Recognition in Surveillance Video". In: *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*. CVPR '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 3153–3160. ISBN: 978-1-4577-0394-2. DOI: 10.1109/CVPR.2011.5995586.

[153] Christopher Olston et al. "TensorFlow-Serving: Flexible, High-performance ML Serving". In: *arXiv preprint arXiv:1712.06139* (2017).

[154] Omid Alipourfard and Hongqiang Harry Liu and Jianshu Chen and Shivaram Venkataraman and Minlan Yu and Ming Zhang. "CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics". In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 2017, pp. 469–482. ISBN: 978-1-931971-37-9. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/alipourfard.

[155] Samuel J Palmisano. "A smarter planet: the next leadership agenda". In: *IBM. November* 6 (2008), pp. 1–8.

[156] Roger Pantos and William May. *HTTP Live Streaming*. 2016. URL: https://tools.ietf.org/html/draft-pantos-http-live-streaming-19.

[157] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. "Deep Face Recognition". In: *Proceedings of the British Machine Vision Conference (BMVC)*. BMVA Press, 2015, pp. 41.1–41.12. ISBN: 1-901725-53-7. DOI: 10.5244/C.29.41.

[158] Qifan Pu et al. "Low Latency Geo-Distributed Data Analytics". In: *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. SIGCOMM '15. London, United Kingdom: ACM, 2015, pp. 421–434. ISBN: 978-1-4503-3542-3. DOI: 10.1145/2785956.2787505.

[159] Jan M Rabaey. "A Brand New Wireless Day". In: *Design Automation Conference, 2008. ASPDAC 2008. Asia and South Pacific*. IEEE. 2008, pp. 1–1.

[160] Ariel Rabkin et al. "Aggregation and Degradation in JetStream: Streaming Analytics in the Wide Area". In: *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*. NSDI'14. Seattle, WA: USENIX Association, 2014, pp. 275–288. ISBN: 978-1-931971-09-6. URL: http://dl.acm.org/citation.cfm?id=2616448.2616474.

[161] Ragunathan Raj Rajkumar et al. "Cyber-Physical Systems: The Next Computing Revolution". In: *Proceedings of the 47th design automation conference*. ACM. 2010, pp. 731–736.

[162] Joseph Redmon. *Darknet: Open Source Neural Networks in C*. http://pjreddie.com/darknet/. 2013–2017.

[163] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, Faster, Stronger". In: *arXiv preprint arXiv:1612.08242* (2016). URL: http://arxiv.org/abs/1612.08242.

[164] Reza Rejaie, Mark Handley, and Deborah Estrin. "Layered Quality Adaptation for Internet Video Streaming". In: *IEEE Journal on Selected Areas in Communications* 18.12 (2000), pp. 2530–2543.

[165] Iain E. Richardson. *The H.264 Advanced Video Compression Standard*. 2nd. Wiley Publishing, 2010. ISBN: 0470516925, 9780470516928.

[166] C. J. Van Rijsbergen. *Information Retrieval*. 2nd. Newton, MA, USA: Butterworth-Heinemann, 1979. ISBN: 0408709294.

[167] Bryan C Russell et al. "LabelMe: a Database and Web-based Tool for Image Annotation". In: *Int. J. Comput. Vision* 77.1-3 (May 2008), pp. 157–173. ISSN: 0920-5691. DOI: 10.1007/s11263-007-0090-8.

[168] Adrian Sampson et al. "EnerJ: Approximate Data Types for Safe and General Low-power Computation". In: *ACM SIGPLAN Notices*. Vol. 46. 6. ACM. 2011, pp. 164–174.

[169] *Samsung SAMI: A Data Exchange Platform that Defines a New Paradigm*. https://developer.samsungsami.io/. 2015.

[170] Mahadev Satyanarayanan. "A Brief History of Cloud Offload: A Personal Journey from Odyssey Through Cyber Foraging to Cloudlets". In: *GetMobile: Mobile Computing and Communications* 18.4 (2015), pp. 19–23.

[171] Mahadev Satyanarayanan et al. "Pervasive computing: Vision and challenges". In: *IEEE Personal communications* 8.4 (2001), pp. 10–17.

[172] Mahadev Satyanarayanan et al. "The Case for VM-based Cloudlets in Mobile Computing". In: *IEEE Pervasive Computing* 8.4 (Oct. 2009), pp. 14–23. ISSN: 1536-1268. DOI: 10.1109/MPRV.2009.82.

[173] Ben Schneiderman and Catherine Plaisant. *Designing the user interface*. 1998.

[174] Philipp Schulz et al. "Latency Critical IoT Applications in 5G: Perspective on the Design of Radio Interface and Network Architecture". In: *IEEE Communications Magazine* 55.2 (2017), pp. 70–78.

[175] H Schulzrinne, A Rao, and R Lanphier. "RTSP: Real time streaming protocol". In: *IETF RFC2326, april* (1998).

[176] H Schulzrinne et al. *RTP: A Transport Protocol for Real-Time*. 2006.

[177] *SFPark*. http://sfpark.org/.

[178] Bobak Shahriari et al. "Taking the human out of the loop: A review of bayesian optimization". In: *Proceedings of the IEEE* 104.1 (2016), pp. 148–175.

[179] Scott Shenker. "Fundamental Design Issues for the Future Internet". In: *IEEE Journal on selected areas in communications* 13.7 (1995), pp. 1176–1188.

[180] Scott Shenker, R Braden, and D Clark. "Integrated services in the Internet architecture: an overview". In: *IETF Request for Comments (RFC)* 1633 (1994).

[181] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[182] *SmartThings*. http://www.smartthings.com/.

[183] Jasper Snoek. *Spearmint*. 2016. URL: https://github.com/HIPS/Spearmint.

[184] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. "Practical Bayesian Optimization of Machine Learning Algorithms". In: *Advances in neural information processing systems*. 2012, pp. 2951–2959.

[185] Iraj Sodagar. "The MPEG-DASH Standard for Multimedia Streaming over the Internet". In: *IEEE MultiMedia* 18.4 (Oct. 2011), pp. 62–67. ISSN: 1070-986X. DOI: 10.1109/MMUL.2011.71.

[186] Benjamin Solnik et al. "Bayesian Optimization for a Better Dessert". In: (2017).

[187] *Spark*. https://www.spark.io/.

[188] Mark Stanislav and Zach Lanier. *The Internet of Fails: Where IoT Has Gone Wrong and How We're Making It Right*. https://www.defcon.org/html/defcon-22/dc-22-speakers.html. 2014.

[189] Wilfried Steiner. "TTEthernet Specification". In: *TTTech Computertechnik AG, Nov* 39 (2008), p. 40.

[190] Yi Sun et al. "CS2P: Improving Video Bitrate Selection and Adaptation with Data-Driven Throughput Prediction". In: *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*. ACM. ACM, 2016, pp. 272–285. DOI: 10.1145/2934872.2934898.

[191] Srikanth Sundaresan et al. "BISmark: A Testbed for Deploying Measurements and Applications in Broadband Access Networks". In: *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*. USENIX ATC'14. Philadelphia, PA: USENIX Association, 2014, pp. 383–394. ISBN: 978-1-931971-10-2. URL: http://dl.acm.org/citation.cfm?id=2643634.2643673.

[192] Melanie Swan. "The Quantified Self: Fundamental Disruption in Big Data Science and Biological Discovery". In: *Big Data* 1.2 (2013), pp. 85–99.

[193] Christian Szegedy et al. "Going Deeper with Convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[194] GStreamer Team. *GStreamer: Open Source Multimedia Framework*. 2001–2017. URL: https://gstreamer.freedesktop.org/.

[195] *Tech Talk: Swarm Boxes*. http://semiengineering.com/tech-talk-swarm-boxes/.

[196] TeleGeography. *Global Internet Geography*. https://www.telegeography.com/research-services/global-internet-geography/. Accessed: 2017-04-10. 2016.

[197] James Temperton. *One nation under CCTV: the future of automated surveillance*. http://www.wired.co.uk/article/one-nation-under-cctv. Accessed: 2017-01-27. 2015.

[198] *The Berkeley Ubiquitous SwarmLab*. https://swarmlab.berkeley.edu/.

[199] *The TerraSwarm Research Center*. https://terraswarm.org/.

[200] Gilman Tolle et al. "A Macroscope in the Redwoods". In: *Proceedings of the 3rd international conference on Embedded networked sensor systems*. ACM. 2005, pp. 51–63.

[201] Ankit Toshniwal et al. "Storm@ twitter". In: *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM. 2014, pp. 147–156. URL: https://dl.acm.org/citation.cfm?id=2595641.

[202] Bobby Vandalore et al. "A Survey of Application Layer Techniques for Adaptive Streaming of Multimedia". In: *Real-Time Imaging* 7.3 (2001), pp. 221–235.

[203] Paul Viola and Michael Jones. "Rapid Object Detection Using a Boosted Cascade of Simple Features". In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. Vol. 1. 2001, I–511–I–518 vol.1. DOI: 10.1109/CVPR.2001.990517.

[204] Raajay Viswanathan, Ganesh Ananthanarayanan, and Aditya Akella. "Clarinet: WAN-Aware Optimization for Analytics Queries". In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. GA: USENIX Association, 2016, pp. 435–450. ISBN: 978-1-931971-33-1. URL: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/viswanathan.

[205] Ashish Vulimiri et al. "Global Analytics in the Face of Bandwidth and Regulatory Constraints". In: *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. 2015, pp. 323–336. URL: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/vulimiri.

[206] Ashish Vulimiri et al. "Low Latency via Redundancy". In: *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*. CoNEXT'13. Santa Barbara, California, USA: ACM, 2013, pp. 283–294. ISBN: 978-1-4503-2101-3. DOI: 10.1145/2535372.2535392.

[207] Ashish Vulimiri et al. "WANalytics: Geo-Distributed Analytics for a Data Intensive World". In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. SIGMOD'15. Melbourne, Victoria, Australia: ACM, 2015, pp. 1087–1092. ISBN: 978-1-4503-2758-9. DOI: 10.1145/2723372.2735365.

[208] Tom Wadlow. *The Questions are the Same, but the Answers are Always Changing*. https://swarmlab.eecs.berkeley.edu/events/2014/11/18/5165/questions-are-same-answers-are-always-changing. 2015.

[209] Gregory K Wallace. "The JPEG Still Picture Compression Standard". In: *Commun. ACM* 34.4 (Apr. 1991), pp. 30–44. ISSN: 0001-0782. DOI: 10.1145/103085.103089.

[210] Bolun Wang et al. "Anatomy of a Personalized Livestreaming System". In: *Proceedings of the 2016 ACM on Internet Measurement Conference*. ACM. 2016, pp. 485–498.

[211] Sheng Wei and Viswanathan Swaminathan. "Low latency live video streaming over HTTP 2.0". In: *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*. ACM. 2014, p. 37.

[212] Mark Weiser. "Ubiquitous Computing". In: *Computer* 26.10 (1993), pp. 71–72.

[213] *Wink*. http://www.wink.com/.

[214] *Xively*. https://xively.com/.

[215] Xiaoqi Yin et al. "A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP". In: SIGCOMM'15 (2015), pp. 325–338. DOI: 10.1145/2785956.2787486.

[216] Madoka Yuriyama and Takayuki Kushida. "Sensor-cloud infrastructure-physical sensor management with virtualized sensors on cloud computing". In: *Network-Based Information Systems (NBiS), 2010 13th International Conference on*. IEEE. 2010, pp. 1–8.

[217] Thomas Zachariah et al. "The Internet of Things Has a Gateway Problem". In: *HotMobile'15*. Santa Fe, New Mexico, USA: ACM, 2015, pp. 27–32. ISBN: 978-1-4503-3391-7. DOI: 10.1145/2699343.2699344.

[218] Matei Zaharia et al. "Discretized Streams: Fault-tolerant Streaming Computation at Scale". In: *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. SOSP'13. Farminton, Pennsylvania: ACM, 2013, pp. 423–438. ISBN: 978-1-4503-2388-8. DOI: 10.1145/2517349.2522737.

[219] Alex Zambelli. "IIS Smooth Streaming Technical Overview". In: *Microsoft Corporation* 3 (2009), p. 40.

[220] Arkady Zaslavsky, Charith Perera, and Dimitrios Georgakopoulos. "Sensing as a Service and Big Data". In: *arXiv preprint arXiv:1301.0159* (2013).

[221] Ben Zhang et al. "AWStream: Adaptive Wide-Area Streaming Analytics". In: *Proceedings of the 2018 ACM Conference on Special Interest Group on Data Communication*. SIGCOMM '18. 2018.

[222] Ben Zhang et al. "HOBS: Head Orientation-based Selection in Physical Spaces". In: *Proceedings of the 2nd ACM symposium on Spatial user interaction*. ACM. 2014, pp. 17–25.

[223] Ben Zhang et al. "The Cloud is Not Enough: Saving IoT from the Cloud." In: *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'15)*. USENIX Association, 2015. URL: https://www.usenix.org/conference/hotcloud15/workshop-program/presentation/zhang.

[224] Haoyu Zhang et al. "Live Video Analytics at Scale with Approximation and Delay-Tolerance". In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 2017, pp. 377–392. ISBN: 978-1-931971-37-9. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/zhang.

[225] P Zhang et al. "Habitat Monitoring with ZebraNet: Design and Experiences". In: *Wireless sensor networks: a systems perspective* (2005), pp. 235–257.

[226] Tan Zhang et al. “The Design and Implementation of a Wireless Video Surveillance System”. In: *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. ACM. 2015, pp. 426–438. URL: https://dl.acm.org/citation.cfm?id=2790123.

[227] Yuyu Zhang et al. “FLASH: Fast Bayesian Optimization for Data Analytic Pipelines”. In: *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD’16. San Francisco, California, USA: ACM, 2016, pp. 2065–2074. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939829.

[228] Feng Zhao and Leonidas Guibas. “Wireless Sensor Networks”. In: *Communications Engineering Desk Reference* 247 (2009).

[229] Xuan Kelvin Zou et al. “Can Accurate Predictions Improve Video Streaming in Cellular Networks?” In: *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*. HotMobile ’15. Santa Fe, New Mexico, USA: ACM, 2015, pp. 57–62. ISBN: 978-1-4503-3391-7. DOI: 10.1145/2699343.2699359.

Appendix A

A List of IoT Hacks

This list of IoT hacks is first created in 2015 to understand the security vulnerabilities in the IoT space. It was made available on Github and received contributions from others.[1]

Table of Contents

- Thingbots
- RFID
- Home Automation
- Connected Doorbell
- Hub
- Smart Coffee
- Wearable
- Smart Plug
- Cameras
- Traffic Lights
- Automobiles
- Airplanes
- Light Bulbs
- Locks
- Smart Scale
- Smart Meters
- Pacemaker
- Thermostats
- Fridge
- Media Player & TV
- Rifle (Weapon)
- Toilet
- Toys

[1] https://github.com/nebgnahz/awesome-iot-hacks/graphs/contributors.

Thingbots

- Proofpoint Uncovers Internet of Things (IoT) Cyberattack

RFID

- Vulnerabilities in First-Generation RFID-enabled Credit Cards
- MIT Subway Hack Paper Published on the Web
- Tampered Card Readers Steal Data via Bluetooth

Home Automation

- IOActive identifies vulnerabilities in Belkin WeMo's Home Automation
- Cameras, Thermostats, and Home Automation Controllers, Hacking 14 IoT Devices
- Popular Home Automation System Backdoored Via Unpatched Flaw

Connected Doorbell

- CVE-2015-4400: Backdoorbot, Network Configuration Leak on a Connected Doorbell

Hub

- TWSL2013-023: Lack of Web and API AuthenticationVulnerability in INSTEON Hub

Smart Coffee

- Reversing the Smarter Coffee IoT Machine Protocol to Make Coffee Using the Terminal

Wearable

- How I hacked my smart bracelet

Smart Plug

- Hacking the D-Link DSP-W215 Smart Plug
- Reverse Engineering the TP-Link HS110
- Hacking Kankun Smart Wifi Plug
- Smart Socket Hack Tutorial

Cameras

- Trendnet Cameras - I always feel like somebody's watching me
- Hacker Hotshots: Eyes on IZON Surveilling IP Camera Security
- Cameras, Thermostats, and Home Automation Controllers, Hacking 14 IoT Devices
- Hacker 'shouts abuse' via Foscam baby monitoring camera

- Urban surveillance camera systems lacking security
- TWSL2014-007: Multiple Vulnerabilities in Y-Cam IP Cameras
- Say Cheese: a snapshot of the massive DDoS attacks coming from IoT cameras
- Samsung SmartCam install.php Remote Root Command Exec

Traffic Lights

- Green Lights Forever: Analyzing The Security of Traffic Infrastructure
- Hacking US (and UK, Australia, France, etc.) Traffic Control Systems

Automobiles

- Hackers Remotely Attack a Jeep on the Highway
- Comprehensive Experimental Analyses of Automotive Attack Surfaces

Airplanes

- Hackers could take control of a plane using in-flight entertainment system

Light Bulbs

- Hacking into Internet Connected Light Bulbs
- Hacking Lightbulbs: Security Evaluation Of The Philips Hue Personal Wireless Lighting System
- IoT Goes Nuclear: Creating a ZigBee Chain Reaction
- Extended Functionality Attacks on IoT Devices: The Case of Smart Lights

Locks

- Lockpicking in the IoT

Smart Scale

- Fitbit Aria Wi-Fi Smart Scale

Smart Meters

- Solar Power Firm Patches Meters Vulnerable to Command Injection Attacks

Pacemaker

- Pacemakers and Implantable Cardiac Defibrillators: Software Radio Attacks and Zero-Power Defenses

Thermostats

- Cameras, Thermostats, and Home Automation Controllers, Hacking 14 IoT Devices
- Google Nest: Exploiting DFU For Root
- Smart Nest Thermostat, A Smart Spy in Your Home
- TWSL2013-022: No Authentication Vulnerability in Radio Thermostat

Fridge

- Proofpoint Uncovers Internet of Things (IoT) Cyberattack - Spam emails from fridges.
- Hacking Defcon 23'S IoT Village Samsung Fridge

Media Player & TV

- Breaking Secure-Boot on the Roku
- Google TV Or: How I Learned to Stop Worrying and Exploit Secure Boot
- Chromecast: Exploiting the Newest Device By Google
- Ransomware Ruins Holiday By Hijacking Family's LG Smart TV on Christmas Day

Rifle (Weapon)

- Hacking a IoT Rifle - BlackHat 2015 - 36 slides
- Hackers Can Disable a Sniper Rifle—Or Change Its Target - Wired 2015

Toilet

- TWSL2013-020: Hard-Coded Bluetooth PIN Vulnerability in LIXIL Satis Toilet

Toys

- TWSL2013-021: Multiple Vulnerabilities in Karotz Smart Rabbit
- Fisher-Price smart bear allowed hacking of children's biographical data (CVE-2015-8269)
- Hello Barbie Initial Security Analysis
- Security researcher Ken Munro discovers vulnerability in Vivid Toy's talking Doll 'Cayla'
- Data from connected CloudPets teddy bears leaked and ransomed, exposing kids' voice messages