Research Statement

Shadi A. Noghabi (abdolla2@illinois.edu)

The focus of my thesis has been on providing *low latency* at *industrial-scale* systems. I treat latency as a first-class citizen in my research. Most of my work is deployed in large-scale and mainstream production systems (e.g., Ambry and Samza) with hundreds of millions of users.

In the era of increased engagement with technology, many highly *interactive* applications processing large amounts of data have emerged. For example, we expect social networks to show us current global and local hashtag trends within seconds, ad campaigns to orient ads based on current user activity, data from IoT (Internet of Things) to be processed within seconds/minutes, and multi-user online gaming to react with milliseconds. In all these diverse areas, handling large scale data in a real-time fashion is a critical need.

At large scale, providing low latency becomes increasingly challenging. Large scale necessitates distributing both processing and storage across large clusters of machines. In many cases, this distribution needs to be spread across the globe for proximity to the data source, while being replicated in multiple locations to handle inevitable failures at scale. Distribution and replication cause load imbalance, scheduling complexities, and excessive use of expensive inter-datacenter traffic. These issues significantly increase latency, especially that they typically impact the critical path of processing.

My research has focused on designing and developing systems that broadly explore these issues with particular attention to improving end-to-end latency and building massive-scale solutions. My research contributions span:

1. **Geo-distributed Storage:** Collaborated with LinkedIn to develop Ambry, a low-latency geo-distributed object store efficient for both small (few KB) and large (few GB) objects [1]. Ambry, including my code contribution, has been the mainstream media storage (photos, documents, slides, etc.) at LinkedIn for over two years.
2. **Stream Processing:** Researched and developed solutions for three major challenges in stream processing [2]: a) managing *large amount of state* (100s of TB in a single job) in a fault-tolerant manner b) *automatically scaling the total resources* of a job [3, 4], and c) *rebalancing resources* among various elements of a job [5]. My work is integrated in two of the most popular Stream Processing frameworks, Apache Storm and Apache Samza. Two of my projects are deployed in LinkedIn's production across over 200 applications, and part of Apache Samza.
3. **Container Networking:** Designed FreeFlow, a container networking solution which achieves both high performance and good portability by leveraging container location information and adding trust among containers belonging to the same application[6]. The team I worked with at Microsoft are analyzing adding this contribution to their production container technology.

1 Geo-distributed Storage

Ambry – Geo-distributed Low Latency Object Storage. For over 24 months, Ambry has been the **mainstream** storage for all LinkedIn's media objects, across all of its four datacenters, serving more than 450 million users. In today's high-tech connected world, an immense amount of data is generated every second from all around the world. Social networks alone generated billions of variable-sized media objects per day. This excessive amount of diverse data needs to be served with low latency from all across the globe, while transparently scaling to match the ever-growing amount of data. In collaboration with LinkedIn, we developed Ambry, a *geo-distributed*, *scalable*, and *low-latency* object store for a *broad range* of objects (a few KBs to a few GBs) [1]. I have been part of the team from the early phases of the project, and have led the effort on rebalancing the system to achieve seamless scalability.

Ambry exploits three main design principals. First, *small overheads are significant for small objects*. Thus, Ambry groups objects into logical units and treats the unit as a single entity. Second, *datacenters should be as*

independent as possible. To achieve this, Ambry serves request in one data center, and later, using a background asynchronous replication mechanism, propagates data to other datacenters. Finally, *load imbalance is inevitable during expansion* (scale out) – new machines are empty while old machines have years-old and unpopular data. Ambry uses a non-intrusive rebalancing strategy based on popularity, access patterns, and size. This strategy uses the spare network bandwidth to move data around in the background.

2 Stream Processing

Stream processing systems provide a useful abstraction: a "stream of infinite data" that should be processed in real-time. The convenience and large scale of these systems have attracted a broad community of users [2]. In my research, I tackled three challenging issues in this area. How to handle *large amount of state* along with processing? How to *automatically scale* resources of a job? How to *distribute and balance resources among a job*? I have developed open-source solutions for these issues, and two of my projects are employed by LinkedIn.

Stateful Stream Processing at Scale. Stateful processing is inherent to many categories of applications, such as large joins, cumulative or compact aggregates, static settings, or machine learning models. As part of my PhD, I worked on *handling a large amount of state efficiently* while *recovering quickly from failures*.

When it comes to large state, relying on an external storage is not a viable solution (orders of magnitude degradation in latency and throughput). Therefore, we developed a state handling mechanism in Apache Samza that leverages the *local storage* of the computing nodes [7]. Our approach partitions state across nodes using the disk as permanent storage and memory as a cache – achieving both the low latency of memory and the large capacity of the disk. Along with state, we use a low-overhead background *changelog mechanism for fault-handling* and speed-up recovery using periodic compaction, parallel recovery, and location aware rescheduling of failed tasks. Our system has successfully been **in production running more than 200 applications** on over 10,000 containers while processing trillions of events per day.

Auto-scaling – Automatic Total Resource Allocation: Finding the ideal resource allocation such that input is not queued, jobs do not fail, and resources are not under-utilized is a sophisticated and time-consuming task. In the case of any workload or logic change, the whole tuning process has to be repeated. Unfortunately, application developers tend to over-provision and waste resources. My contribution has been to automate this configuration by iteratively scaling (adding or removing) resources until a near optimal allocation is reached [3]. As an intern at LinkedIn, I integrated an three phase feedback loop auto-scaler (profiling, triggering, and scaling) into Apache Samza. Exploiting queuing theory and bin-packing, the auto-scaler detects over- and under-utilized jobs and scales them accordingly. The results of my work are employed by LinkedIn.

Rebalancing – Redistributing Resources within a Job: Along with deciding the total resources, another exciting challenge is *how to allocate resources among the various elements of a job?* Stream processing jobs are formed as a DAG of operators, with a fixed amount of resources distributed across the DAG. Poor resource distribution creates bottlenecks, which in turn slow down the whole pipeline. During my PhD, I developed a queuing theory based rebalancing strategy on top of Apache Storm [5]. This approach adaptively redistributes resources among operators until all bottlenecks are resolved even in the presence of workload changes.

3 Container Networking

FreeFlow – High Performance Cross Container Networking. During an internship at Microsoft Research, I worked on FreeFlow [6], a container networking solution which achieves both *high performance* and *good portability*. Containerization has piqued a strong interest in Big Data Analytics particularly because of its high portability and low overhead – containers have no external dependencies while having an independent and isolated namespace. However, current container networking solutions have either poor performance or portability, undermining the advantages of containerization. To fill this gap, we designed and developed FreeFlow, a high performance and portable container networking solution.

FreeFlow leverages two insights: first, in most container deployments a central entity (i.e. the orchestrator) exists that is fully aware of the *location of each container*. Second, *strict isolation is unnecessary* among containers belonging to the same application, e.g., containers of a single MapReduce job. Leveraging these two observations,

FreeFlow uses a variety of technologies such as shared memory and RDMA to improve network performance (higher throughput, lower latency, and less CPU overhead), while maintaining full portability and does all this in a manner that is completely transparent to application developers.

4 Future Research

Edge Computing (or so called the "Edge") is becoming a leading technology, opening the venue for a wide range of interesting applications ranging from self-driving cars, smart cities/homes, to wireless Virtual and Augmented Reality. In the future, I plan to build on my experience in large scale low-latency system development towards *building a sustainable easy-to-use end-to-end framework for Edge computing*. This avenue provides an interesting combination of multidisciplinary design and system building problems. I believe I can have a noticeable and lasting impact on this emerging area of Big Data.

Managing Millions of Geo-distributed Edges. Reaching the vision of "Edge-everwhere" – having the Edge available in any location – opens an important research avenue of *how to monitor, manage, and revive Edges at large scale?* Many interesting questions need to be answered: How to scale management to hundreds of thousands of Edges? How to provide high availability and fault tolerance? How to recover from the common human errors in such an environment (e.g., unplugging the device)? How to handle a dead Edge that needs to be replaced?

What makes these issues particularly interesting in Edge computing is *geo-distribution*, *heterogeneity* of devices, and the *unreliability* of the environment. The network to/from Edges is an unreliable, (usually) expensive, and low-bandwidth network. Finally, the Edge devices are faulty and diverse with various hardware configurations/capacities.

Multi-tenancy at the Edge. In a setting with *many limited-resource Edges* and *several diverse jobs*, how should applications be scheduled? Edge computing has a unique resource distribution pattern. Edges are heterogeneous, fast (near the data source), and with limited resources and bandwidth. On the other hand, the Cloud is a complimentary option, with homogeneous high-resource high(er)-latency resources. On this platform, a broad mix of latency sensitive, resource/bandwidth heavy, energy limited, and privacy constrained applications need to run concurrently.

This unique setting opens an exciting avenue on how to scheduling diverse multi-tenant jobs, with many questions to address. For a given application, which part(s) should run on the close-by while limited-resource Edge and which on the Cloud? Which jobs should be prioritized to use the Edge? When a new job comes in, how should the system react? How to minimize the impact of jobs on one another? How to schedule a broad mix of jobs to increase utilization without hurting performance? During the internship at Microsoft Research, I started investigating how to partition a job across an Edge and a Cloud. The focus of this work was building a scheduling mechanism that hides the heterogeneity at the Edges while being scalable. I plan to continue researching in this area to build a pluggable end-to-end solution.

Media Processing Using the Edge. Given the *communication heavy*, *computation heavy*, and *latency sensitive* nature of media processing, *how can the Edge be used to facilitate media processing*? Media has become dominant in almost all aspects of our lives, from medical, military and security to social media. However, processing media requires special treatment mainly because it is both communication heavy (large objects) and computation heavy (complex processing) [8]. Moreover, in many use cases timeliness is also crucial, e.g., Virtual Reality and self-driving cars. These requirements make using the Cloud unpractical, either because of latency or the high bandwidth required. My plan is to *build low latency media processing environments by leveraging the Edge*. The goal is to create an easy-to-use framework that automatically detects latency sensitive and resource sensitive operations in an application and transparently partitions the application across the close by Edges and the Cloud.

References

[1] **Shadi A. Noghabi**, Sriram Subramanian, Priyesh Narayanan, Sivabalan Narayanan, Gopalakrishna Holla, Mammad Zadeh, Tianwei Li, Indranil Gupta, and Roy H. Campbell. Ambry: Linkedin's scalable geo-

distributed object store. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 253–265. ACM, 2016.

[2] **Shadi A. Noghabi**, Le Xu, Indranil Gupta, and Roy H. Campbell. Building blocks of large scale stream processing: the past, current, and future. *(in progress) to be submitted to ACM Computing Surveys (CSUR)*, 2017.

[3] **Shadi A. Noghabi**. Auto-scaling containers in samza. `https://issues.apache.org/jira/browse/SAMZA-719`, .

[4] **Shadi A. Noghabi**. Evaluating auto-scaling in samza. `https://issues.apache.org/jira/browse/SAMZA-755`, .

[5] **Shadi A. Noghabi**, Read Sprabery, John Bellessa, Mohammad Ahmad, Indranil Gupta, and Roy H. Campbell. Real time adaptive profiling in storm topologies. *Technical Report*, 2015.

[6] Tianlong Yu, **Shadi A. Noghabi**, Shachar Raindel, Hongqiang Liu, Jitu Padhye, and Vyas Sekar. Freeflow: High performance container networking. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, HotNets '16, pages 43–49. ACM, 2016.

[7] **Shadi A. Noghabi**, Kartik Paramasivam, Yi Pan, Navina Ramesh, Jon Bringhurst, Indranil Gupta, and Roy H. Campbell. Samza: Stateful stream processing at scale. *Proceedings of the VLDB*, 2017.

[8] **Shadi A. Noghabi**, Roy H. Campbell, and Indranil Gupta. Building a scalable distributed online media processing environment. In *Proceedings of the VLDB (PhD Workshop)*, 2016.