Building Interactive Distributed Processing Applications at a Global Scale

Shadi A. Noghabi

Prelim Committee: Prof. Roy Campbell (advisor), Prof. Indy Gupta (advisor), Prof. Klara Nahrstedt, Dr. Victor Bahl
Many Interactive Applications

- < few seconds
- 100s of PBs data
- < few minutes
- Trillions of events
- < few milliseconds
- Millions - billions of devices
Low Latency at Large Scale

- **Latency-sensitive**, reaching *low latency* is an important factor for interactivity
  
  100ms latency (Amazon) $\rightarrow$ 1% loss in revenue $\rightarrow$ $4M per millisec! *

- **Large scale** processing, at large and global scales

My focus: Building Interactive systems at a production scale

* https://blog.gigaspaces.com/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/
At Scale, Low Latency is Challenging

**Heterogeneity**
- Data
- Machines
- Operations

**Dynamism**
- Workload
- Environment

**Distribution**

**State & Replication**

**Imbalance**
Latency-driven designs can be used to build production infrastructures for interactive applications at a global scale while addressing myriad challenges on heterogeneity, dynamism, state, and imbalance.
Latency-driven Design

Achieving **latency** has the **highest priority**

PACELC $^{1,2}$

- Latency
- Consistency
- Availability

- Latency
- Throughput
- Isolation

Related Work

Latency-Driven in other frameworks
• Trend of NoSQL systems: Cassandra, Amazon’s DynamoDB, Pileus.

Latency-Driven does not fit all
• Consistency driven:
  – Consistent Storage: BigTable, Spanner, Yahoo’s PNUTS, Espresso
  – Exactly-once Stream processing: Flink, Millwheel, Spark Streaming, Trident
• Throughput driven:
  – Batch processing: Hadoop, Spark, Tez, Pig and Hive
  – Micro-batch streaming: Spark Streaming, media processing
  – High-throughput Storage: HDFS and GFS
Latency-driven Techniques

- Background processing
- Tiered data access
- Load balancing
- Locality
- Partitioning & parallelism
- Adaptive Scaling
- Opportunistic processing
Latency-driven at All Layers

General container networking applicable to Docker, Kubernetes, etc.
Latency-driven Techniques

- Background processing
- Tiered data access
- Load balancing
- Locality
- Partitioning & parallelism
- Adaptive Scaling
- Opportunistic processing

Ambry [SIGMOD’16]
Samza [VLDB’17]
Steel [HotCloud’18]
FreeFlow [HotNets’17]
Latency-driven Techniques

- **Background processing**
- **Tiered data access**
- **Load balancing**
- **Locality**
- **Partitioning & parallelism**
- **Adaptive Scaling**
- **Opportunistic processing**

---

**Storage**
- Ambry [SIGMOD’16]
- Samza [VLDB’17]
- Steel [HotCloud’18]
- FreeFlow [HotNets’17]

---

**Networking**

---

**Processing**
Samza: Stateful Scalable Stream Processing at LinkedIn

VLDB’17

Shadi A. Noghabi*, Kartik Paramasivam^, Yi Pan^, Navina Ramesh^, Jon Bringhurst^, Indranil Gupta*, Roy Campbell*

* University of Illinois at Urbana-Champaign
^ LinkedIn Corp.
Stream Processing*

*Stream processing is the processing of data in motion, i.e., computing on data immediately as it is produced.

- Interactive feeds or ads
- Security & Monitoring
- Internet of Things
Stream Applications need State

State: durable data read/written along with the processing

Many cases need large state

• **Aggregation:** aggregating counts over a window
• **Join:** two (or more) streams over a window
• **Other:** machine learning model
Scale of Processing

Large scale of **input**, in Kafka alone:
- **2.1 Trillion** msg/Day, 16 Million msg/sec peaks
- **0.5 PB in, 2 PB out** per day

Many **applications**:
- **100s** of production applications
- **Over 10,000** containers

Large scale of **state**
- **Several 100s of TBs** for a single application
Stateful Stream Processing

Need to handle state with low latency, at large-scale, and with correct (consistent) results

- Opensource Apache project
- Powers hundreds of apps in LinkedIn’s production
- In use at LinkedIn, Uber, Metamarkets, Netflix, Intuit, TripAdvisor, VMware, Optimizely, Redfin, etc.

http://samza.apache.org/
Stateful Stream Processing

Processing Model
- Input partitioning
- Parallel and independent tasks
- Locality aware data passing

Stateful
- Local state
- Incremental checkpointing
- 3-Tier caching
- Parallel recovery
- Host Stickiness
- Compaction
Stateful Stream Processing

**Processing Model**
- Input partitioning
- Parallel and independent tasks
- Locality aware data passing

**Stateful**
- Local state
- Incremental checkpointing
- 3-Tier caching
- Parallel recovery
- Host Stickiness
- Compaction
Processing from a Partitioned Source

Stream of events

Ex: page access, ad views, etc.
Processing from a Partitioned Source

Stream of events

Ex: page access, ad views, etc.

Partition 1

Partition 2

... Partition k

20
Processing from a Partitioned Source

Stream of events

Ex: page access, ad views, etc.

Partition 1

Partition 2

... Partition k
Processing from a Partitioned Source

Stream of events

Ex: page access, ad views, etc.

Partition 1

Partition 2

Partition k

Task 1

Task 2

Task k

Partitioning & parallelism
Multi-Stage Dataflow

Application logic: Count number of ‘Page accesses’ for each ip domain in a 5 minute window

Data Parallelism: each task performs the entire pipeline on its own chunk of input data
Stateful Stream Processing

Processing Model

• Input partitioning
• Parallel and independent tasks
• Locality aware data passing

Stateful

• Local state
• Incremental checkpointing
• 3-Tier caching
• Parallel recovery
• Host Stickiness
• Compaction
Stateful Stream Processing

Processing Model
- Input partitioning
- Parallel and independent tasks
- Locality aware data passing

Stateful
- Local state
- Incremental checkpointing
- 3-Tier caching
- Parallel recovery
- Host Stickiness
- Compaction
Stateful Processing

Store count of access per ip domain

But, using a remote store is slow
Stateful Processing

Samza Application

Task 1
Task 2
...
Task k

State: Ip access count

[1-100]
[900-1000]
## Stateful Processing

**Local State:** use the local storage of tasks
- Each task, state corresponding to their partition.
- Ex: task 1, processing input [1-100], state [1-100]
Stateful Processing

Local Storage:

- **in-memory**: fast, but low capacity (GBs)
- **on-disk**: high capacity (TBs) and persistent, but slow
- **Disk-mem**: Disk (persistent store) + memory (cache)
  - Fast, high capacity, and persistent

Tiered data access
Stateful Processing

Samza Application

- Task 1
- Task 2
- Task $k$

[1-100] [100-200] [900-1000]

What about failures?
How to not lose state?
Fault-Tolerance

Changes saved to a durable change log

→ Recovery by replay change-log

Periodically batch & flush changes

Changelog

e.g., Kafka log compacted topic

partition k

partition 2

partition 1
Fault-Tolerance

Changes saved to a durable change log

→ Recovery by replay change-log

Periodically batch & flush changes

Changelog
e.g., Kafka log compacted topic

partition 1
partition 2
partition k
Fault-Tolerance

Changes saved to a durable change log

→ Recovery by replay change-log

Periodically batch & flush changes
Fault-Tolerance

Changes saved to a durable change log

→ Recovery by replay change-log

Periodically batch & flush changes

Changelog
  e.g., Kafka log compacted topic

Background processing

X = 10
But, what is the cost?
But, what is the cost?

Latency

Consistency

Availability

Replayable input + Consistent snapshot

**Longer** failure recovery
Consistency Guarantees

Offsets
e.g., Kafka topic

Offset 1005

Changelog
e.g., Kafka topic

Offset 1005

Task 1
Task 2
... 
Task k

partition k partition 2 partition 1

X = 10
Available Optimizations

- Availability is compromised. To improve we use:
  - Parallel recovery
  - Background compaction
  - Host stickiness
Fast Restarts with Host Stickiness

Input Stream

Task 1, Task 4 -> Host-A
Task 2 -> Host-B
Task 3 -> Host-C

Job

Task-1
Host-A

Task-4

Task-2
Host-B

Task-3
Host-C

Change-log

Durable:
Task-Container-Host Mapping

Host Stickiness in YARN:
- Try to place task on same host after restart
- Minimize state rebuilding Overhead
Related Work

• **Stateless:** do *not support state*
  – Apache Storm [SIGMOD’14], Heron [SIGMOD’15], S4 [ICDM’10]

• **Remote Store:** relying on external storage
  – Trident, MillWheel [VLDB’13], Dataflow [VLDB’15]
  – High latency overhead per input message
  – *but*, fast recovery (better availability)

• Local state, but, full state **checkpointing:**
  – Flink [TCDE’15], Spark Streaming [HotCloud’12], IBM Streams [VLDB’16], Streamscope [NDSI’16], System S [DEBS’11]
  – Does *not scale* for large application state
  – *but*, easier “repeatable results” at failure
Evaluation
Evaluation Setup

• Production Cluster
  – 500 node YARN cluster
  – real world applications

• Small Cluster
  – 8 node cluster: 64GB RAM, 24 core CPUs, a 1.6 TB SSD
  – micro-benchmarks
    • Read-only workload ~ join with table
    • Read-write workload ~ aggregation over time
Local State -- Latency

Stores:
- in-mem
- on-disk
- disk-mem
- remote

Fault-tolerance:
- None
- changelog (Clog)
- remote

> 2 orders of magnitude slower compared to local state

changelog adds minimal overhead
on disk w/ caching comparable with in memory
Failure Recovery

- Growing linearly with size of state
- Almost constant overhead with host stickiness
- Overhead independent of % of failures (parallel recovery)
# Summary of State in Samza

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>• 100x better latency</td>
<td>• Dependent to input partitioning</td>
</tr>
<tr>
<td>• Better resource utilization</td>
<td>• Does NOT work when state is large and not co-partitionable in input stream</td>
</tr>
<tr>
<td>• no need for remote DB resources</td>
<td></td>
</tr>
<tr>
<td>• Parallel and independent processing</td>
<td>• Auto-scaling becomes harder</td>
</tr>
<tr>
<td></td>
<td>• Recovery is slower (vs remote)</td>
</tr>
</tbody>
</table>
Conclusion

Need to handle *state* with *low latency*, at *large-scale*, and with *correct (consistent)* results

**Processing Model**
- Input partitioning
- Parallel and independent tasks
- Locality aware data passing

**Stateful**
- Local state
- Incremental checkpointing
- 3-Tier caching
- Parallel recovery
- Host Stickiness
- Compaction

**Locality**
**Background processing**
**Partitioning & parallelism**
Latency-driven Techniques

- Background processing
- Tiered data access
- Load balancing
- Locality
- Partitioning & parallelism
- Adaptive Scaling
- Opportunistic processing

Storage
- Ambry [SIGMOD’16]
- Samza [VLDB’17]
- Steel [HotCloud’18]
- FreeFlow [HotNets’17]
Latency-driven Techniques

- Background processing
- Tiered data access
- Load balancing
- Locality
- Partitioning & parallelism
- Adaptive Scaling
- Opportunistic processing

Ambry [SIGMOD’16]
Samza [VLDB’17]
Steel [HotCloud’18]
FreeFlow [HotNets’17]
Steel: Unified and Optimized Edge-Cloud Environment

*Shadi A. Noghabi*, John Kolb^, Peter Bodik**, Eduardo Cuervo**, Indranil Gupta*, Roy Campbell*

* University of Illinois at Urbana-Champaign
^ University of California, Berkeley
** Microsoft Research
Cloud is Not an One-Size-Fits-All

Latency

< 80ms          < 20ms          < 10 ms

The Cloud is simply too far!
> 70 ms round trip time
Cloud is Not an One-Size-Fits-All

Latency

< 80ms
< 20ms
< 10 ms

Resources

Bandwidth
10s – 100s GB/s

Battery

Energy

Heating

Not enough resources to use the Cloud
Cloud is Not an One-Size-Fits-All

Latency

< 80ms
< 10 ms

Resources

Bandwidth
10s – 100s GB/s

Battery
Energy
Heating

Privacy & Security

…
However...

Industry is at its **infancy** of building **Edge-Cloud applications**

- **No proper unification** of Edge & Cloud
  - Hard to **configure, deploy, and monitor**
  - *Manual* and **redundant** optimizations

**Steel**: A **unified** edge cloud framework with **modular** and **automated optimizations**

Integrated with *production* Azure services
Steel

Applications

Abstraction (Logical Spec)

Fabric

Compile → Deploy → Monitor & Analyze

Optimization Modules

Placement → Communication → Load Balancing

Edge-Cloud Ecosystem
Optimizer Modules

### Placement
Where (edge/cloud) to place?
Adapt to **long-term** changes

- **Location** and **vicinity** aware
- Background **profiling** & what-if analysis
- Background **batching & compression**
- **Partitioned** and parallel optimizations
- **Change** strategy **opportunistically** (available resources)

### Communication
configure edge-cloud links
Adapt to **short-term** spikes

- **Locality**
- **Partitioning & parallelism**
- **Background processing**
- **Opportunistic processing**
Latency-driven Techniques

Background processing
Tiered data access
Load balancing
Locality
Partitioning & parallelism
Adaptive Scaling
Opportunistic processing

Storage
Ambry
Samza
Steel
FreeFlow

Networking
Processing

SIGMOD'16
VLDB'17
HotCloud'18
HotNets'17

Networking
Processing
Latency-driven Techniques

- Background processing
- Tiered data access
- Load balancing
- Locality
- Partitioning & parallelism
- Adaptive Scaling
- Opportunistic processing

Ambry [SIGMOD’16]
Samza [VLDB’17]
Steel [HotCloud’18]
FreeFlow [HotNets’17]
Ambry: LinkedIn’s Scalable Geo-Distributed Object Store

SIGMOD ‘16

Shadi A. Noghabi*, Sriram Subramanian+, Priyesh Narayanan+, Sivabalan Narayanan+, Gopalakrishna Holla+, Mammad Zadeh+, Tianwei Li+, Indranil Gupta*, Roy H. Campbell*

* University of Illinois at Urbana-Champaign
SRG: http://srg.cs.illinois.edu and DPRG: http://dprg.cs.uiuc.edu
+ LinkedIn Corporation
Data Infrastructure team
Massive Media Objects are Everywhere

100s of Millions of users

100s of TBs to PBs

From all around the world
How to handle all these objects?

We need a **geographically distributed** system that stores and retrieves objects in an **low-latency** and **scalable** manner.

**Ambry**

In production for **>4 years** and **>500 M** users!

Open source: [https://github.com/linkedin/ambry](https://github.com/linkedin/ambry)
Geo-Distribution

Asynchronous writes
- Write synchronously only to local datacenter
- Asynchronously replicate others
- Reduce user perceived latency
- Minimize cross-DC traffic
Geo-Distribution

Asynchronous writes
- Write synchronously only to local datacenter
- Asynchronously replicate others
- Reduce user perceived latency
- Minimize cross-DC traffic
Ambry is a large production system with many other techniques:

- Logical **data partitioning**
- **Caching**, indexing, bloom filters
- Background **load balancing**
- ....
Latency-driven Techniques

- Background processing
- Tiered data access
- Load balancing
- Locality
- Partitioning & parallelism
- Adaptive Scaling
- Opportunistic processing

Storage
- Ambry [SIGMOD'16]
- Samza [VLDB'17]
- Steel [HotCloud'18]
- FreeFlow [HotNets'17]

Networking

Processing

Networking
Latency-driven Techniques

Latency Driven

- Background processing
- Tiered data access
- Load balancing
- Locality
- Partitioning & parallelism
- Adaptive Scaling
- Opportunistic processing

Storage
- Ambry [SIGMOD’16]
- Samza [VLDB’17]
- Steel [HotCloud’18]
- FreeFlow [HotNets’17]
FreeFlow: High performance container networking

HotNets ‘16

Tianlong Yu^, Shadi A. Noghabi *, Shachar Raindel†, Hongqiang Liu†, Jitu Padhye†, and Vyas Sekar^

* University of Illinois at Urbana-Champaign
† Microsoft Research
^ Carnegie Mellon University
Containers are popular

Containers provide good **portability** and **isolation**

[Image of logos: Docker, CoreOS, Kubernetes, Mesos]

[Chart: Does your organization run container technologies? Answered: 297]

79% Yes, 19% No, 2% Don’t Know

FreeFlow

Node 1
- Container 1
- Shared memory

Node 2
- Container 3

RDMA

Opportunistic processing

Locality

Network
Latency-driven Techniques

- Background processing
- Tiered data access
- Load balancing
- Locality
- Partitioning & parallelism
- Adaptive Scaling
- Opportunistic processing

Storage: Ambry [SIGMOD’16]
- Samza [VLDB’17]
- Steel [HotCloud’18]
- FreeFlow [HotNets’17]
Future Directions

• *One global system* seamlessly across the globe
  – *Holistic* and *cross-layer* approach
• Latency *sensitivity-aware* approaches (not treating all objects/apps equally)
  – E.g., dynamic use of compression, erasure coding, compaction in storage for old objects
  – E.g., multi-tenant stream processing environments
• *Auto-pilot systems* coping with changes automatically and dynamically
Thanks to Many Collaborators

• **Advisors:** Indranil Gupta and Roy Campbell

• **LinkedIn:** Sriram Subramanian, Priyesh Narayanan, Kartik Paramasivam, Yi Pan, Navina Ramesh, et al.

• **Microsoft Research:** Victor Bahl, Peter Bodik, Eduardo Cuervo, Hongqiang Liu, Jitu Padhye, et. al.
Publications

2. “Steel: Simplified Development and Deployment of Edge-Cloud Applications”, SA Noghabi, J Kolb, P Bodik, E Cuervo, HotCloud 2018
7. “To edge or not to edge?”, F Kalim, SA Noghabi, S Verma, SoCC poster 2017
8. “Building a Scalable Distributed Online Media Processing Environment”, SA Noghabi, PhD workshop at VLDB 2016
Latency-driven designs can be used to build production infrastructures for interactive applications at a global scale.
Back up Slides
Latency-driven design can be modeled theoretically with **Queueing Theory**

**System:** A hierarchical queuing system

**Latency:** Response time \((response\_processed - req\_enter\_time)\)

**Techniques:** queueing theory models/techniques
- **Parallelism & Partitioning** → multiple queues, parallel servers
- **Adaptive scaling** → adding more servers
- ...

Other Techniques

**Batch + Stream** in one System

- **Reprocess** parts/all stream or Database
  - bugs, upgrades, logic changes

→ **Batch as a finite stream**
  + conflict resolution, scaling & throttling
Local State -- Throughput

Remote state 30-150x worse than local state

On disk w/ caching comparable with in memory

Changelog adds minimal overhead
Snapshot vs Changelog