Typhoon: An SDN Enhanced Real-Time Big Data Streaming Framework

Junguk Cho, Hyunseok Chang, Sarit Mukherjee, T.V. Lakshman, and Jacobus Van der Merwe
Big Data Era

- Big data analysis is increasingly common
  - Recommendation systems (e.g., Netflix, Amazon)
  - Targeted advertising (e.g., Google, Facebook)
  - Mobile network management (e.g., AT&T, Verizon)

- Real time stream framework
  - Continuously process unlimited streams of data
  - High throughput and low latency using computation parallelism
  - Fault tolerance
  - A lot of open-source stream frameworks (e.g., Storm, Heron, Flink, Spark etc.)
Limitations of Current Real-Time Stream Frameworks

- Lack of runtime flexibility
  - Scale-in/out, computation logic, and routing policy
Limitations of Current Real-Time Stream Frameworks

● Lack of runtime flexibility
  ○ Scale-in/out, computation logic, and routing policy
  ○ Solutions
    ■ Shutdown -> modification -> restart
      ● Lose data
Limitations of Current Real-Time Stream Frameworks

● Lack of runtime flexibility
  ○ Scale-in/out, computation logic, and routing policy
  ○ Solutions
    ■ Shutdown -> modification -> restart
      ● Lose data
    ■ Instant swapping
      ● Require stable front-end queue systems (e.g., kafka)
      ● Still resource inefficient
Limitations of Current Real-Time Stream Frameworks

● Lack of runtime flexibility
  ○ Scale-in/out, computation logic, and routing policy
  ○ Solutions
    ■ Shutdown -> modification -> restart
      ● Lose data
    ■ Instant swapping
      ● Require stable front-end queue systems (e.g., kafka)
      ● Still resource inefficient

● Not optimal for one-to-many communication
  ○ No broadcast concept
  ○ Multiple same computations in application-layer
Limitations of Current Real-Time Stream Frameworks

- Lack of runtime flexibility
  - Scale-in/out, computation logic, and routing policy
  - Solutions
    - Shutdown -> modification -> restart
      - Lose data
    - Instant swapping
      - Require stable front-end queue systems (e.g., kafka)
      - Still resource inefficient

- Not optimal for one-to-many communication
  - No broadcast concept
  - Multiple same computations in application-layer

Problem on inflexibility and performance is mainly from application-layer data routing
Limitations of Current Real-Time Stream Frameworks

● Lack of runtime flexibility
  ○ Scale-in/out, computation logic, and routing policy
  ○ Solutions
    ■ Shutdown -> modification -> restart
      • Lose data
    ■ Instant swapping
      • Require stable front-end queue systems (e.g., kafka)
      • Still resource inefficient

● Not optimal for one-to-many communication
  ○ No broadcast concept
  ○ Multiple same computations in application-layer

● Lack of management for deployed stream apps
  ○ Limited to application monitoring
Typhoon: An SDN Enhanced Real-Time Stream Framework

- Offer flexible stream processing pipelines for dynamic reconfigurations
  - Scale-in/out, computation logic, routing policy
- Optimize one-to-many communication
  - Broadcast concept at network layer
- Enhance the management capabilities in stream frameworks
  - SDN control plane applications

**Cross-layer design**
- Partially offload application data routing layer into network layer
- Manage both layers from an SDN controller
Why SDN in Stream Framework?

- **SDN benefits**
  - Centralized views and control
  - Programmability

- **One-to-many communication**
  - More efficient packet-level replications at the network layer

- **Well-defined protocol and interface**
  - OpenFlow

- **Evolvable framework**
  - Extensible via control plane applications
Can We Apply SDN to Stream Framework?

- Conceptually similar to computer networks

Graph-based communication pattern
Can We Apply SDN to Stream Framework?

- Conceptually similar to computer networks

![Graph-based communication pattern](image)

![Computing & Routing components](image)
Can We Apply SDN to Stream Framework?

- All topology information pre-defined in submitted application
Can We Apply SDN to Stream Framework?

- All topology information pre-defined in submitted application

Well fit in proactive SDN model

Stream frameworks

App Developer

New Application Or Reconfiguration Request
Current Stream Frameworks

Streaming Manager
- Topology Builder & Scheduler

Central Coordinator

Worker Agent
- Worker
- ..... Worker

Compute Cluster
Typhoon Design Challenges

- How can we integrate SDN into stream frameworks?
- How can we support dynamic reconfigurations?
Typhoon Design Challenges

- How can we integrate SDN into stream framework?
- How can we support dynamic reconfigurations?
Integration of SDN into Stream Framework
Integration of SDN into Stream Framework

Worker Agent

Worker

Software SDN Switch

Worker

Data Computation

Data Routing

I/O Layer

Worker Agent

Worker

Software SDN Switch
Integration of SDN into Stream Framework

Worker Agent

Worker

Software SDN Switch

Worker Agent

Worker

Data Computation

Data Routing

I/O Layer

Worker

Data Computation

Data Routing

I/O Layer

Port

Software SDN Switch

Port
To use SDN as data forwarding functions, Typhoon needs match fields in flow rules and packet format to be matched.
**Data Tuple:** stream communication data model for worker-to-worker communications in existing stream frameworks

<table>
<thead>
<tr>
<th>Metadata of data tuple</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuple Length</td>
</tr>
<tr>
<td>Destination worker ID</td>
</tr>
<tr>
<td>Source worker ID</td>
</tr>
<tr>
<td>StreamID</td>
</tr>
<tr>
<td>Output from Data Computation</td>
</tr>
</tbody>
</table>
Data Tuple: stream communication data model for worker-to-worker communications in existing stream frameworks

Unique IDs in a stream application

<table>
<thead>
<tr>
<th>Tuple Length</th>
<th>Destination worker ID</th>
<th>Source worker ID</th>
<th>StreamID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metadata of data tuple</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Output from Data Computation
**Data Tuple:** stream communication data model for worker-to-worker communications in existing stream frameworks

**Metadata of data tuple**

- Tuple Length
- Destination worker ID
- Source worker ID
- StreamID

**Unique IDs in a stream application**

**Custom transport protocol in Typhoon**

<table>
<thead>
<tr>
<th>Ethernet header</th>
<th>Destination worker ID</th>
<th>Source worker ID</th>
<th>Ether type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Payload (Data tuple)</td>
<td>Tuple Length</td>
<td>StreamID</td>
<td></td>
</tr>
<tr>
<td>Output from Data Computation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Metadata of data tuple in Typhoon
Offloading Data Forwarding from Application Layer

Packet format created by I/O Layer

<table>
<thead>
<tr>
<th>Ethernet header</th>
<th>Destination worker ID</th>
<th>Source worker ID</th>
<th>Ether type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Payload</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Source worker:
  - Data Computation
  - Data Routing
  - I/O Layer

- Destination worker:
  - Data Computation
  - Data Routing
  - I/O Layer

Pre-installed flow rule

<table>
<thead>
<tr>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_port=[src worker port], dl_dst=[dest worker ID], dl_src=[src worker ID]</td>
<td>output=[dest worker port]</td>
</tr>
</tbody>
</table>

Software SDN Switch
Typhoon Design Challenges

- How can we integrate SDN into stream framework?
- **How can we support dynamic reconfigurations?**
  - What are the requirements for dynamic reconfigurations?
Worker data routing
- Policy-specific routing states
- Policy-independent routing states
- Routing computations

/* Round robin */
Index = (counter++) % numDestWorkers;
dstWorker = nextWorkers[index];
Application Layer Data Routing in Typhoon

Worker

- Update routing states if needed
  - Policy-independent routing states
    - Scale-in/out
    - Computation logic update
  - Policy-specific routing states
    - Change routing type and states

<table>
<thead>
<tr>
<th>Routing Type</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>Round-robin</td>
<td>Counter</td>
</tr>
<tr>
<td>Key-based</td>
<td>Index for hash</td>
</tr>
<tr>
<td>One-to-many</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Destination Worker IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination 1</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Destination N</td>
</tr>
</tbody>
</table>
Typhoon Design Challenges

- How can we integrate SDN into stream framework?
- **How can we support dynamic reconfigurations?**
  - What are the requirements for dynamic reconfigurations?
    - Updating per-worker routing states for reconfigurations
Typhoon Design Challenges

- How can we integrate SDN into stream framework?
- **How can we support dynamic reconfigurations?**
  - What are the requirements for dynamic reconfigurations?
    - Updating per-worker routing states for reconfigurations
    - Framework-level coordination to avoid packet loss during reconfiguration
Typhoon Design Challenges

- How can we integrate SDN into stream framework?
- **How can we support dynamic reconfigurations?**
  - What are the requirements for dynamic reconfigurations?
    - Updating per-worker routing states for reconfigurations
    - Framework-level coordination to avoid packet loss during reconfiguration
  - How can Typhoon manage per-worker routing states in centralized manners?
  - What are the protocol and Interface for managing application-layer routing states?
  - How can Typhoon prevent packet loss during reconfigurations?
How to manage per-worker routing states in Typhoon

- Offloading routing states management from the application layer to the network layer

- SDN Controller (i.e., network)
  - Manage application-layer routing states with centralized manners
    - Policy-independent routing states (e.g., forwarding tables)
    - Policy-specific routing states (e.g., index for key-based routing)
What is a management layer in Typhoon?

- What is a **transport method** to control per-worker?
  - PacketOut in OpenFlow (i.e., Protocol)
  - Each worker is coupled to a port in a software SDN switch.
What is a management layer in Typhoon?

- What is a transport method to control per-worker?
  - PacketOut in OpenFlow (i.e., Protocol)
  - Each worker is coupled to a port in a software SDN switch.

- What is a message format to carry routing states?
  - Control tuple: Re-use data tuple format in stream frameworks for framework compatibility (i.e., Interface)
Stable Dynamic Reconfiguration in Typhoon

- How can Typhoon prevent packet loss during reconfigurations?
  - Well-defined order to update each component under framework coordination
Stable Dynamic Reconfiguration in Typhoon

Control tuple $\rightarrow$ SDN flow rules $\rightarrow$ Added worker $\rightarrow$ SDN controller

Source $\rightarrow$ New destination $\rightarrow$ Scale-out $\rightarrow$ Scale-in

Destination
Stable Dynamic Reconfiguration in Typhoon

No packet loss under framework coordination
Runtime Flexibility in Typhoon

● Design flexible stream processing pipelines
  ○ Offload routing managements to an SDN controller
    ■ Leverage control tuples carried by OpenFlow PacketOut
    ○ Offload the data forwarding to network with SDN flow rules

● Support reconfigurations under framework coordination
  ○ Prevent packet loss during reconfigurations
Limitations of Current Real-Time Stream Frameworks

- Lack of runtime flexibility
- **Not optimal for one-to-many communication**
- Lack of management for deployed stream apps
Suboptimal One-to-Many Communication

Source Worker

Data Computation

Data Routing

I/O Layer

Destination Worker 1

Destination Worker 2

Output from Data Computation
## Suboptimal One-to-Many Communication

### Data Tuple

<table>
<thead>
<tr>
<th>Tuple Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination worker ID</td>
</tr>
<tr>
<td>Source worker ID</td>
</tr>
<tr>
<td>StreamID</td>
</tr>
<tr>
<td>Output from Data Computation</td>
</tr>
</tbody>
</table>

### Diagram

- **Source Worker**
  - Data Computation
  - Data Routing
  - I/O Layer

- **Destination Worker 1**
  - Tuple Length
  - Destination worker ID
  - Source worker ID
  - StreamID
  - Output from Data Computation

- **Destination Worker 2**
  - Tuple Length
  - Destination worker ID
  - Source worker ID
  - StreamID
  - Output from Data Computation
Suboptimal One-to-Many Communication

- Degrade performance due to multiple same computations
  - Multiple same computations
    - Heavy multiple serializations
    - Data copy to different TCP connections
  - No broadcast concept in application layers
Typhoon for One-to-Many Communication

Source Worker

Data Computation

Data Routing

I/O Layer

Destination Worker 1

Data Computation

Data Routing

I/O Layer

Destination Worker 2

Data Computation

Data Routing

I/O Layer

Port

Port

Port

Match

in_port=[source worker port], dl_dst=[BROADCAST], dl_src=[source worker ID]

Action

output=[all destination worker’s ports]

Software SDN Switch
Typhoon for One-to-Many Communication

<table>
<thead>
<tr>
<th>Ethernet header</th>
<th>Source Worker ID</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BROADCAST</strong></td>
<td>0xffff</td>
<td>Set of tuples</td>
</tr>
</tbody>
</table>

**Source Worker**

- Data Computation
- Data Routing
- I/O Layer
- Port

**Destination Worker 1**

- Data Computation
- Data Routing
- I/O Layer
- Port

**Destination Worker 2**

- Data Computation
- Data Routing
- I/O Layer
- Port

**Match**

- in_port=[source worker port], dl_dst=[BROADCAST], dl_src=[source worker ID]

**Action**

- output=[all destination worker’s ports]

Software SDN Switch
## Typhoon for One-to-Many Communication

**Ethernet header**

<table>
<thead>
<tr>
<th>Data Routing</th>
<th>Source Worker ID</th>
<th>0xffff</th>
</tr>
</thead>
</table>

**Payload**

| Set of tuples |

---

**Source Worker**

- **Data Computation**
- **Data Routing**
- **I/O Layer**

**Destination Worker 1**

- **Data Computation**
- **Data Routing**
- **I/O Layer**

**Destination Worker 2**

- **Data Computation**
- **Data Routing**
- **I/O Layer**

**Match**

\[
\text{in\_port}=\text{[source worker port]}, \quad \text{dl\_dst}=\text{[BROADCAST]}, \\
\text{dl\_src}=\text{[source worker ID]}
\]

**Action**

\[
\text{output}=\text{[all destination worker’s ports]}
\]

**Software SDN Switch**
One-to-Many Communication In Typhoon

- **Straightforward, but very effective**
- Use a broadcast concept in network
  - One-to-many communication aware worker I/O and an SDN controller
  - Packet-level replications at the network layer
Limitations of Current Real-Time Stream Frameworks

- Lack of runtime flexibility
- Not optimal for one-to-many communication
- Lack of management for deployed stream apps
Limitations of Current Real-Time Stream Frameworks

- Lack of runtime flexibility
- Not optimal for one-to-many communication
- Lack of management for deployed stream apps

Only monitoring

### Topology stats

<table>
<thead>
<tr>
<th>Window</th>
<th>Emitted</th>
<th>Transferred</th>
<th>Complete latency (ms)</th>
<th>Acked</th>
<th>Failed</th>
</tr>
</thead>
<tbody>
<tr>
<td>10m 0s</td>
<td>69166860</td>
<td>69166660</td>
<td>1274.317</td>
<td>17018600</td>
<td>0</td>
</tr>
<tr>
<td>3h 0m 0s</td>
<td>69166860</td>
<td>69166660</td>
<td>1274.317</td>
<td>17018600</td>
<td>0</td>
</tr>
<tr>
<td>1d 0h 0m 0s</td>
<td>69166860</td>
<td>69166660</td>
<td>1274.317</td>
<td>17018600</td>
<td>0</td>
</tr>
<tr>
<td>All time</td>
<td>69166860</td>
<td>69166660</td>
<td>1274.317</td>
<td>17018600</td>
<td>0</td>
</tr>
</tbody>
</table>

### Spouts (All time)

<table>
<thead>
<tr>
<th>Id</th>
<th>Executors</th>
<th>Tasks</th>
<th>Emitted</th>
<th>Transferred</th>
<th>Complete latency (ms)</th>
<th>Acked</th>
<th>Failed</th>
</tr>
</thead>
<tbody>
<tr>
<td>kafka-spout</td>
<td>10</td>
<td>10</td>
<td>34677140</td>
<td>34677120</td>
<td>1274.317</td>
<td>17018600</td>
<td>0</td>
</tr>
</tbody>
</table>

Showing 1 to 1 of 1 entries

### Bolts (All time)

<table>
<thead>
<tr>
<th>Id</th>
<th>Executors</th>
<th>Tasks</th>
<th>Emitted</th>
<th>Transferred</th>
<th>Capacity (last 10m)</th>
<th>Execute latency (ms)</th>
<th>Executed</th>
<th>Process latency (ms)</th>
<th>Acked</th>
<th>Failed</th>
<th>Error Host</th>
<th>Error Port</th>
<th>Last error</th>
</tr>
</thead>
<tbody>
<tr>
<td>__acker</td>
<td>10</td>
<td>10</td>
<td>17151000</td>
<td>17150980</td>
<td>0.307</td>
<td>0.006</td>
<td>34302000</td>
<td>0.006</td>
<td>34301980</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>es-bolt</td>
<td>60</td>
<td>60</td>
<td>17338720</td>
<td>17338560</td>
<td>0.218</td>
<td>0.017</td>
<td>17338520</td>
<td>0.013</td>
<td>17338580</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SDN Control Plane Applications

- **Unique opportunities by tightly coupling framework and SDN**
  - Rich cross-layer information from application and network layers

- **SDN control plane applications based on cross-layer information**
  - Fault detector
  - Auto Scaler
  - Live debugger
  - Load balancer
Fault Detector SDN Control Plane Application

- Existing stream frameworks
  - Polling approach
  - Heartbeat based on timeout
    - Expensive in case of large scale
    - Not instantaneous

- Typhoon
  - Event-driven approach and network layer information
  - OpenFlow event
    - Unexpected port deletion events
    - Traffic steering with flow rule changes
Auto Scaler SDN Control Plane Application

- How to decide scale-in/out operation
  - Application-layer information is more accurate than network information to know worker is overloaded
    - Capacity = (# of processed tuples * average process time) / measurement time

- Auto Scaler SDN control plane application
  - Threshold-based scale-in/out operations using capacity metric (i.e., application layer information)
Typhoon Prototype: Refactor Apache Storm

- **Re-define global states using Thrift**
  - Update global states to include reconfigurations and detailed topology information stored in a central coordinator (i.e., zookeeper)

- **Stream manager (i.e., Nimbus)**
  - Extend the topology builder and the scheduler
  - Implement dynamic topology manager module
    - For updating global states for initial submissions and reconfigurations

- **Worker I/O layer**
  - Replace netty-based Storm transport with Typhoon custom transport
  - Implement with JAVA and C using JNI to bridge them
    - C using the DPDK framework
      - (De)Packetizing and Packet batching in C
    - Java implementing Storm transport interface
      - Queue and batch tuples
      - (De)Multiplexing tuples
Typhoon Prototype

- **Software SDN switch**
  - DPDK-based user space OVS

- **Floodlight controller**
  - Use apache curator for interaction between an SDN controller and the zookeeper
  - Implement SDN control plane applications
  - Provide rest APIs for control plane applications
  - Re-use tuple libraries from Storm for control tuples
Evaluation

- Baseline performance
- One-to-many communication performance
- SDN control plane applications
  - Auto Scaler
  - Fault detector
  - Update compute nodes
  - Live debugger
  - Load balancer
Evaluation

- Baseline performance
- One-to-many communication performance
- SDN control plane applications
  - Auto Scaler
  - Fault detector
  - Update compute nodes
  - Live debugger
  - Load balancer
One-to-Many Communication Performance

Increase the number of destination workers from two to six
One-to-Many Communication Performance

Increase the number of destination workers from two to six

Source

Destination

Destination

# of destination workers

<table>
<thead>
<tr>
<th># of destination workers</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

# Million Tuples/sec

STORM (LOCAL)  |
| TYPHOON (LOCAL) |
| STORM (REMOTE) |
| TYPHOON (REMOTE) |
One-to-Many Communication Performance

Increase the number of destination workers from two to six

Typhoon outperforms Storm due to lightweight packet replications in the SDN switch
Auto Scaler SDN Control Plane Application

Packet loss due to faulty split worker
Auto Scaler SDN Control Plane Application

Scale-out split worker

Input

Split

Count

Split

Count

Split

Count

- Scale

(b) Throughput in count workers.

(c) Throughput in split workers.
Auto Scaler SDN Control Plane Application

Scale-out split worker

Typhoon avoids packet loss due to worker faults using threshold-based auto scaler
Conclusion

- Propose an SDN-based real time stream framework which tightly integrates SDN functionality into a realtime stream frameworks

- Demonstrate that the proposed system provides highly flexible stream pipeline and high-performance application-level data broadcast using cross-layer design and SDN

- Showcase several SDN control plane applications leveraging cross-layer information