SAND: A Fault-Tolerant Streaming Architecture for Network Traffic Analytics

Qin Liu, John C.S. Lui
Cheng He, Lujia Pan, Wei Fan, Yunlong Shi

1The Chinese University of Hong Kong
2Huawei Noah’s Ark Lab
Introduction
Motivation

Network traffic arrives in a **streaming fashion**, and should be processed **in real-time**. For example,

1. Network traffic classification
2. Anomaly detection
3. Policy and charging control in cellular networks
4. Recommendations based on user behaviors
Challenges

1. A stream processing system must sustain high-speed network traffic in cellular core networks
   - existing systems: S4 [Neumeyer’10], Storm \(^1\) ...
   - implemented in Java: heavy processing overheads
   - cannot sustain high-speed network traffic

\(^1\)http://storm.incubator.apache.org/
Challenges

1. A stream processing system must **sustain high-speed network traffic** in cellular core networks
   - existing systems: S4 [Neumeyer’10], Storm \(^1\) ...
   - implemented in Java: heavy processing overheads
   - cannot sustain high-speed network traffic

2. For critical applications, it is necessary to **provide correct results after failure recovery**
   - high hardware cost
   - cannot provide “correct results” after failure recovery
   - at-least-once vs. exactly-once

\(^1\)http://storm.incubator.apache.org/
Contributions

Design and implement SAND in C++:
- high performance on network traffic
- a new fault tolerance scheme
Background
Background

Continuous operator model:
- Each node runs an operator with in-memory mutable state
- For each input event, state is updated and new events are sent out

Mutable state is lost if node fails.
Example: AppTracker

- AppTracker: traffic classification for cellular network traffic
- Output traffic distribution in real-time:

<table>
<thead>
<tr>
<th>Application</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP</td>
<td>15.60%</td>
</tr>
<tr>
<td>Sina Weibo</td>
<td>4.13%</td>
</tr>
<tr>
<td>QQ</td>
<td>2.56%</td>
</tr>
<tr>
<td>DNS</td>
<td>2.34%</td>
</tr>
<tr>
<td>HTTP in QQ</td>
<td>2.17%</td>
</tr>
</tbody>
</table>
Example: AppTracker

Under the continuous operator model:

- **Spout**: capture packets from cellular network
- **Decoder**: extract IP packets from raw packets
- **DPI-Engine**: perform *deep packet inspection* on packets
- **Tracker**: track the distribution of application level protocols (HTTP, P2P, Skype ...)
System Design
Architecture of SAND

One *coordinator* and multiple *workers*. Each worker can be seen as an operator.
Coordinator

Coordinator is responsible for

- managing worker executions
- detecting worker failures
- relaying control messages among workers
- monitoring performance statistics

Zookeeper cluster provides fault tolerance and reliable coordination service.
Worker

Contain 3 types of processes:

- The **dispatcher** decodes streams and distributes them to multiple analyzers
- Each **analyzer** independently processes the assigned streams
- The **collector** aggregates the intermediate results from all analyzers

The **container** daemon

- spawns or stops the processes
- communicates with the coordinator
Communication Channels

Efficient communication channels:

- Intra-worker: a lock-free shared memory ring buffer
- Inter-worker: ZeroMQ, a socket library optimized for clustered products
Fault-Tolerance
Previous Fault-Tolerance Schemes

1. **Replication**: each operator has a replica operator [Hwang’05, Shah’04, Balazinska’08]

   - Data streams are processed twice by two identical nodes
   - Synchronization protocols ensure exact ordering of events in both nodes
   - On failure, the system switches over to the replica nodes

   **2x hardware cost.**
Previous Fault-Tolerance Schemes

2. *Upstream backup with checkpoint* [Fernandez’03, Gu’09]:
   - Each node maintains backup of the forwarded events since last checkpoint
   - On failure, upstream nodes replay the backup events serially to the failover node to recreate the state

Less hardware cost. It’s hard to provide **correct results** after recovery.
Why is it hard?

- Stateful continuous operators tightly integrate “computation” with “mutable state”
- Makes it harder to define clear boundaries when computation and state can be moved around
Checkpointing

- Need to coordinate checkpointing operation on each worker
- 1985: Chandy-Lamport invented an asynchronous snapshot algorithm for distributed systems
- A variant algorithm was implemented within SAND
Checkpointing Protocol

- Coordinator initiates a global checkpoint by sending markers to all source workers
- For each worker $w$,
  - on receiving a data event $E$ from worker $u$
    - if marker from $u$ has arrived, $w$ buffers $E$
    - else $w$ processes $E$ normally
  - on receiving a marker from worker $u$
    - if all markers have arrived, $w$ starts checkpointing operation
Checkpointing Operation

On each worker:

- When a checkpoint starts, the worker creates child processes using fork
- The parent processes then resume with the normal processing
- The child processes write the internal state to HDFS, which performs replication for data reliability
Output Buffer

Buffer output events for recovery:

- Each worker records output data events in its output buffer, so as to replay output events during failure recovery
- When global checkpoint $c$ is finished, data in output buffers before checkpoint $c$ can be deleted
Failure Recovery

- $F$: failed workers
- $D_F$: downstream workers of $F$
- $F \cup D_F$: rolled back to the most recent checkpoint $c$
- $P_F$: the upstream workers of $F \cup D_F$
- Workers in $P_F$ replay output events after checkpoint $c$
Evaluation
**Experiment 1**

- Testbed: one quad-core machine with 4GB RAM
- Dataset: packet header trace; 331 million packets accounting for 143GB of traffic
- Application: packet counter

<table>
<thead>
<tr>
<th>System</th>
<th>Packets/s</th>
<th>Payload Rate</th>
<th>Header Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storm</td>
<td>260K</td>
<td>840Mb/s</td>
<td>81.15Mb/s</td>
</tr>
<tr>
<td>Blockmon</td>
<td>2.7M</td>
<td>8.4Gb/s</td>
<td>844.9Mb/s</td>
</tr>
<tr>
<td>SAND</td>
<td>9.6M</td>
<td>31.4Gb/s</td>
<td>3031.7Mb/s</td>
</tr>
</tbody>
</table>

- 3.7X and 37.4X compared to Blockmon [Simoncelli’13] and Storm
Experiment 2

- **Testbed**: three 16-core machines with 94GB RAM
- **Dataset**: a 2-hour network trace (32GB) collected from a commercial GPRS core network in China in 2013
- **Application**: AppTracker
Experiment 2

- Scale out by running parallel workers on multiple servers
- Negligible overheads
Experiment 3

- Recover in order of seconds
- Recovery time is in proportion to checkpointing interval
Conclusion

- Present a new distributed stream processing system for network analytics
- Propose a novel checkpointing protocol that provides reliable fault tolerance for stream processing systems
- SAND can operate at core routers level and can recover from failure in order of seconds
Thank you!

Q & A