StreamHub: A Massively Parallel Architecture for High-Performance Content-Based Publish/Subscribe

Raphaël Barazzutti, Pascal Felber, Christof Fetzer, Emanuel Onica, Jean-François Pineau, Marcelo Pasin, Etienne Rivière, Stefan Weigert

University of Neuchâtel, Switzerland
TU Dresden, Germany

DEBS 2013, Arlington, TX
etienne.riviere@unine.ch
- Event-based composition of distributed applications
- Running on several administrative domains
- Content-based publish/subscribe as a Service
- Deployed on a public cloud (or dedicated cluster)
- Communication based on subscriptions on the content of publications
- Requirements?
• **Throughput**
  - Store thousands *subscriptions*
  - Filter thousands *publications* per second
  - Dispatch thousands to millions *notifications* per second

• **Low and consistent delays**

• **Scalability**

• **Support arbitrary filtering schemes**
Support Arbitrary Filtering Schemes

- **Attribute-based** filtering scheme widely studied in literature
  - Represent content using a set of attributes
  - Subscriptions = conjunctions of discrete predicates on attributes values
  - Broker overlays typically rely on *containment & aggregation* capabilities of attribute-based filtering

- Alternative/novel filtering schemes
  - Encrypted filtering
    - ASPE (Choi et al., DEXA10)
    - Prefiltering (DEBS 2012)
  - Any processing on publications’ payload
    - In both case, no guaranteed support for containment or aggregation

- The design of the pub/sub engine cannot always depend on the characteristics of the filtering scheme(s) it supports
StreamHub principles

- **Tiered approach** to pub/sub for cloud/cluster deployment
  - Split pub/sub into three fundamental, consecutive operations
  - Exploit massive data parallelism of each operation

- Event flows do not depend on a filtering scheme characteristics
  - Filtering libraries treated as black boxes

- **StreamHub core =** Stream Processing application
  - Each pub/sub operation mapped to an operator
  - DAG of operators supported by a Stream Processing Engine
Stream Processing Engine: assumptions

- Possible platforms: S4, Storm, StreamMine, ...
  - Externalized state management, no state sharing between slices
  - Need support for unicast, anycast & broadcast primitives
The StreamHub Engine

<table>
<thead>
<tr>
<th>Operator</th>
<th>Access Point (AP)</th>
<th>Matching (M)</th>
<th>Exit Point (EP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role</td>
<td>Subscriptions Partitioning</td>
<td>Publications Filtering</td>
<td>Publications Dispatching</td>
</tr>
</tbody>
</table>

- **Operator**
  - **Access Point (AP)**: Decide where to store subs, Broadcast pubs to next operator
  - **Matching (M)**: Store subs, Filter incoming pubs, Forward list of matching subscribers ids for next operator
  - **Exit Point (EP)**: Aggregate lists of matching subscribers ids, Prepare & dispatch notifications

Streams and Connections:
- DCCP: Persistent TCP connections
- Unicast, Anycast: broadcasting methods
Events Paths and Support Libraries

Determining key to matching operator using clustering

Optional

Libcluster

Store subscription in operator slice state

Libfilter

Subscriber

TCP

Acast

Unicast

DCCP

AP

Static component

StreamHub client

Libraries:
- Operator slice
- State
- Library

Static component

StreamHub client
Events Paths and Support Libraries

**libcluster**

- Determine key to matching operator using clustering

**libfilter**

- Filter publication, return matching subscribers list
- Store subscription in operator slice state

**Publisher**

- TCP
- Anycast

**Subscriber**

- TCP
- Anycast

**Optional**

- Operator slice
- State
- Library
- Static component

StreamHub client
Filtering Libraries

- **Simple API**
  - `void register(subscription)`
  - `void unregister(subscription identifier)`
  - `list<subscribers identifiers> filter(publication)`

- Libfilter attached to one M operator
  - Multiple filtering schemes
    - multiple M operators

- In our evaluation: counting algorithm
  - *a la* SIENA
  - Efficient attribute-based filtering

- Also supported: encrypted filtering
  - ASPE mechanism
  - *Not in present paper*
Clustering Libraries

- Some filtering schemes benefit from **subscription clustering**
  - Similar subscriptions \( \Rightarrow \)
    - More efficient filtering data structures \( \Rightarrow \) Higher throughput
- **libcluster** library paired with a **libfilter**
- Subscription insertion time increased, publication filtering time decreased
- Two clustering libraries for counting **libfilter**
  - **K-Means**
  - **Event Space Partitioning**
Evaluation
Experimental setup

- 48 nodes/384 cores cluster, 1 Gbps switched network
- 48 x Intel Xeon 2 GHz, each with 8 GB RAM
- 32 nodes for StreamHub
- 16 nodes for workload generator / sinks
- Use of **batching** between operators (up to 16 KB)
- Metrics of interest
  - **Throughput** (subscription storage, publication filtering)
  - **Scalability**
  - **Delays**
  - In the paper: impact of using a clustering library
Workload

- Publications: 5 years of quotes from Yahoo! finance for 200 randomly selected stocks (250,000 publications)
- Subscriptions: synthetic subscriptions, based on Meghdoot [Middleware'04] evaluation categories
- We use 100,000 subscriptions unless explicitly mentioned
- 1 publication match a median of 0.18% subscriptions (180 notifications)
Storing Subscriptions

- 8 generator nodes, subscriptions-only workload
- 8 AP nodes (not a bottleneck), EP nodes not used

![Graph showing input throughput (MB/s) vs. number of M operator slices]

35.4 MB/s = 150,000 subscriptions stored per second
Operator-per-operator Evaluation

- 100,000 stored subscriptions, saturating system with publications
- Evaluate each operator scalability by replacing next operator by a sink
- **Main results**
  - The AP-M broadcast is not a bottleneck
  - All operators show linear horizontal scalability
  - Notification throughput is linearly proportional to # stored subscriptions
Throughput and Scalability

- **8 to 32 machines** supporting the engine, 100,000 subscriptions
- **8 generators sending publications**, 16 sinks receiving notifications
- Allocation of servers between AP, M and EP based on operator-by-operator evaluation

4.26x speedup
(better allocation, less contention)

Nearly **400,000 notifications** sent per second

Scalability: contribution / node
Notification delays

| Configuration | (AP 4|M 4|EP 8) | (AP 8|M 8|EP 16) |
|---------------|----------|--------------|
| Batching      | 16 K     | 16 K         |
| Publications/s | 500      | 1,000        |
| Average delay | 1.06 s   | 0.98 s       |
| Std. dev.     | 0.28 s   | 0.3 s        |

• Delays measured from generation of publication on source and reception of notifications on sinks
  • For sources and sinks on the same node (same clock)
  • Throughput is 50% to 57% of maximal throughput

• Delays are small and predictable
  • Using batching impacts delays, but improves throughput
Conclusion

• StreamHub: massively parallel pub/sub engine
  • High throughput and low delays
  • Tailored for cluster and cloud deployments
    • Pub/sub as a Service for application composition
  • Tiered approach for scalability and performance
  • Built as a stream processing engine application

• Extension: elastic scaling of the StreamHub engine
StreamHub: A Massively Parallel Architecture for High-Performance Content-Based Publish/Subscribe

Questions?

This work was supported by the EU FP7 program (SRT-15 project)
http://www.srt-15.eu

DEBS 2013, Arlington, TX
etienne.riviere@unine.ch