Elastic Scaling of a High-Throughput Content-Based Publish/Subscribe Engine

Raphaël Barazzutti, Thomas Heinze, André Martin, Emanuel Onica, Pascal Felber, Christof Fetzer, Zbigniew Jerzak, Marcelo Pasin and Etienne Rivière

University of Neuchâtel, Switzerland
SAP AG, Germany
TU Dresden, Germany

ICDCS 2014, Madrid, Spain
etienne.riviere@unine.ch
Context

• Publish/Subscribe as a service
  • Running on private or public clouds
  • Decoupled communication
  • Composition of applications from multiple domains

• Content-based filtering
  • Subscriptions filter on the content of publications
  • Canonical example: stock quotes filtering

Publisher

Subscriber

Adm. domain A

Adm. domain B

Adm. domain C

name = "IBM"
price > 120$
volume > 10,000

name = "IBM"
price = 131$
volume = 12,312

name = "IBM"
price = 112$
volume = 9,892
open = 109$
close = 113$

Sunday 6 July 14
Pub/sub as a service: requirements

- **Arbitrary representation** of publications and subscriptions
  - Not limited to attribute-based filtering
  - Untrusted domains: encrypted filtering

- **High-throughput and scalability**
  - Thousands subscriptions per second
  - Thousands publications per second
  - Thousands to millions notifications per second
  - Availability, dependability and low delays

- **StreamHub [DEBS 2013]**
  - Supports arbitrary filtering scheme, in particular encrypted filtering (ASPE)
  - Built on top of a Stream Processing Engine

---

Sunday 6 July 14
Problem: resource provisioning

- Pub/sub service resource usage is unpredictable and varies over time
  - Example: Frankfurt stock exchange, Nov 18, 2011
- Throughput/delay requirements vs budget require appropriate provisioning
- This talk: can we make a high-throughput pub/sub service elastic?
  - Scale in and scale out based on actual requirements
  - Maintain service continuity and minimize reconfiguration impact

![Graph showing ticks per second over time]
Outline

• **Background**
  - StreamHub high-throughput content-based pub/sub engine
  - StreamMine3G stream processing engine

• **(elastic) e-StreamHub principles**
  - Components
  - Slice migration
  - Enforcing elasticity policies

• **Evaluation**
  - Micro-benchmarks
  - Trace-based using Frankfurt stock exchange workload
StreamHub: principles

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

- Supports arbitrary filtering mechanism
  - Event flows do not depend on a filtering scheme characteristics
  - This work: use computationally intensive encrypted filtering

- StreamHub engine = Stream Processing application
  - Each of the 3 pub/sub operations mapped to an stream operator
  - Operators supported by a Stream Processing Engine
StreamMine3G stream processing engine

- Distributed stream computation as a DAG of operators
- Each operator split in multiple slices
  - Externalized state management, no state sharing between slices
- Support for unicast, anycast & broadcast primitives
StreamHub Engine [DEBS13]

Operator | Access Point (AP) | Matching (M) | Exit Point (EP) |
--- | --- | --- | --- |
Function | Subscriptions Partitioning | Publications Filtering | Publications Dispatching |
State | Stateless | Stateful (persistent) | Stateful (transient) |
Role | ➤ Decide where to store subs | ➤ Store subs | ➤ Aggregate lists of matching subscribers ids |
| ➤ Broadcast pubs to next operator | ➤ Filter incoming pubs, create list of matching subscribers ids | ➤ Prepare & dispatch notifications |

Publisher

Subscribers

x > 3 & y == 5

x = 7

y = 10

encrypts

C5F80 BA363

88F3B 2A09C

stores

encrypts

DCCP/Connection Point

DCCP/Connection Point

Public Cloud deployment

AP:1

AP:2

AP:3

AP:4

AP:5

AP:6

M:1

M:2

M:3

M:4

M:5

M:6

EP:1

EP:2

EP:3

EP:4

EP:5

EP:6

Anycast

Unicast

Multicast

broadcast (pubs)

Unicast (subs)

DCCP (same)

ASPE

ICDCS 2014 - Etienne Rivière
e(elastic)-StreamHub principle

- Fixed number of slices for each operator
- Elastic scaling decisions based on experienced load
  - Scale out: allocate new host(s), migrate slices
  - Scale in: migrate slices, deallocate host(s)
  - Redistribute slices upon load unbalance between hosts
e-StreamHub: components

- **StreamMine3G**: support for slice migration (including state)
  - Goal: minimal interruption of flow
- **Manager**: orchestrates migration and collect workload probes
- **SM3G** and manager state stored in **ZooKeeper** for dependability
- **Elasticity enforcer**: takes migration decisions based on observed load
  - Based on elasticity policies

[Diagram showing the components of e-StreamHub]

Sunday 6 July 14
1. **Input events duplicated** to a buffer on *dest* host
2. **Slice halted; state is copied** from *src* to *dest* host
   - Incoming events are still buffered
   - States associated with vectors of sequence numbers, one for each of previous operator slices
   - Allows knowing which events were accounted in
3. **Missed events replayed** on state, slice resumed
   ➡ “Downtime” of the slice is essentially **equivalent to state copying time**
Elasticity policy

- **CPU utilization probes** collected periodically
  - Secondary metrics: network usage and memory usage
- **Global rules**: criteria on the average load
  - Trigger *scale-in* and *scale-out* operations
  - Define a target average CPU utilization
- **Local rules**: criteria on the load of a single host
- **Grace period** between migrations (30s)
- **Minimize cost** of migrations (sum of migrated states)

<table>
<thead>
<tr>
<th>Rules</th>
<th>CPU</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>average &gt; 70%</td>
<td>scale out</td>
</tr>
<tr>
<td></td>
<td>average &lt; 30%</td>
<td>scale in</td>
</tr>
<tr>
<td>Local</td>
<td>&lt; 20% or &gt; 80%</td>
<td>redistribute slices</td>
</tr>
<tr>
<td>Target</td>
<td>average CPU = 50%</td>
<td></td>
</tr>
</tbody>
</table>
Elasticity policy enforcement

Three-steps resolution when global or local rule is violated

1. Decide on the set of slices to migrate
   - For each host, identify a set of slices with:
     \[ \text{sum}(\text{CPU utilization}) \geq \text{abs}(\text{current average utilization} - \text{target utilization}) \]
   - *Subset sum* problem: dynamic programming, pseudo-polynomial complexity
   - Returns multiple solutions: select the one with less state transfer involved

2. Scaling out: add enough hosts for \( \text{avg(load)} \) to be \( \leq \) target (50%)
   Scaling in: mark enough least-loaded hosts for \( \text{avg(load)} \) to be \( \geq \) target (50%)

3. Decide on new placement
   - First-fit bin packing algorithm
   - Start with current state without selected slices, greedily assign by decreasing weight
Evaluation
Experimental setup

- Prototype in C, C++ and Java
- 22 hosts, each with 8 cores and 8 GB
  - 1 Gbps switched network
- 4 hosts with G and S: generators and sinks
- 3 hosts for infrastructure
- 1 to 15 hosts used for e-StreamHub
  - 8 to 120 cores
- Number of slices is fixed
  - micro-benchmarks: 4 AP, 8 M, 4 EP
  - trace-based evaluation: 8 AP, 16 M, 8 EP
- Encrypted pubs and subs (ASPE)
  - 100,000 encrypted subs
  - Each sub is ~1.2 KB
  - 1% matching ratio: 1,000 notifications / pub
Static e-StreamHub: Performance

Maximum throughput

Distribution of delays under half of max. throughput

Publications/s

Delays (ms)

Number of hosts and attribution to operators (AP|M|EP)

Per second:
42.2 million matchings
422,000 notifications

Median delay

I2 hosts:
3 hosts for AP slices
6 hosts for M slices
3 hosts for EP slices

ICDCS 2014 - Etienne Rivière

Sunday 6 July 14
Slice migration: times

- Individual migration times, averaged over 25 migrations
- Under constant flow of 100 publications / second
- Worst-case scenario with “large” slices states
  - 4 AP, 8 M and 4 EP with 100K and 400K subscriptions
- Migration time minimal for AP and EP, dominated by state size for M

<table>
<thead>
<tr>
<th></th>
<th>AP</th>
<th>M (12.500 subscriptions, total of 100,000)</th>
<th>M (50.000 subscriptions, total of 400,000)</th>
<th>EP</th>
</tr>
</thead>
<tbody>
<tr>
<td>average</td>
<td>232 ms</td>
<td>1.497 s</td>
<td>2.533 s</td>
<td>275 ms</td>
</tr>
<tr>
<td>standard deviation</td>
<td>31 ms</td>
<td>354 ms</td>
<td>1.557 s</td>
<td>52 ms</td>
</tr>
</tbody>
</table>
Impact of migrations on delay

migrations: AP (1) AP (2) M (1) M (2) EP

delay (seconds)
time (seconds)

min average max

Sunday 6 July 14
Steady increase/decrease of load

- Increase/decrease of publication rate
  - 0 to 350 publications per second
  - 0 to 35 millions matching operations per second

- CPU load
  - average, max and min over 30 seconds
Steady increase/decrease of load

- **Increase/decrease of publication rate**
  - 0 to 350 publications *per second*
  - 0 to 35 millions matching operations *per second*

- **CPU load**
  - average, max and min over 30 seconds
Steady increase/decrease of load

- **Increase/decrease of publication rate**
  - 0 to 350 publications \textit{per second}
  - 0 to 35 millions matching operations \textit{per second}

- **CPU load**
  - average, max and min over 30 seconds

- **Notification delays**
  - average over 30 seconds
Frankfurt stock exchange

- Replay of the Frankfurt stock exchange workload
  - Sped up 10 times: about 40 minutes shown
  - 100,000 encrypted subscriptions

- CPU load
  - average, max and min over 30 seconds
Frankfurt stock exchange

- Replay of the Frankfurt stock exchange workload
  - Sped up 10 times: about 40 minutes shown
  - 100,000 encrypted subscriptions

- CPU load
  - average, max and min over 30 seconds

- Notification delays
  - average over 30 seconds
Conclusion

• Elastic scaling of a high-throughput content-based pub/sub engine
  • Implemented at the level of the support stream processing engine
  • Applicability to other long-lived stream processing applications
  • Support of live operator slice migration
  • Redistribution of slices according to monitored workload

• Allows deploying pub/sub as a service on a public cloud without the need to provision for worst-case scenarios

• Evaluation using Frankfurt stock exchange traces

• Perspectives
  • Monitor network flows, minimize inter-host communications in migration plans
  • Leverage active replication, used for dependability, to minimize impact on delay while migrating (this requires deterministic execution)
Elastic Scaling of a High-Throughput Content-Based Publish/Subscribe Engine

Questions?

This work was supported by the EU FP7 program (SRT-15 project)

http://www.srt-15.eu

ICDCS 2014, Madrid, Spain
etienne.riviere@unine.ch
Additional slides
Independence from Filtering Scheme

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

- Alternative filtering schemes
  - Encrypted filtering
    - ASPE (Choi et al., DEXA10)
    - Prefiltering (DEBS 2012)
  - No guaranteed support for containment or aggregation

- The architecture of a pub/sub engine should be independent from the filtering scheme(s) it supports
Independence from Filtering Scheme

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

- Alternative filtering schemes
  - Encrypted filtering
    - ASPE (Choi et al., DEXA10)
    - Prefiltering (DEBS 2012)
  - No guaranteed support for containment or aggregation

- The architecture of a pub/sub engine should be independent from the filtering scheme(s) it supports
Independence from Filtering Scheme

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

- Alternative filtering schemes
  - Encrypted filtering
    - ASPE (Choi et al., DEXA10)
    - Prefiltering (DEBS 2012)
  - No guaranteed support for containment or aggregation

- The architecture of a pub/sub engine should be independent from the filtering scheme(s) it supports
Independence from Filtering Scheme

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

- Alternative filtering schemes
  - Encrypted filtering
    - ASPE (Choi et al., DEXA10)
    - Prefiltering (DEBS 2012)
  - No guaranteed support for containment or aggregation

- The architecture of a pub/sub engine should be independent from the filtering scheme(s) it supports
Events Paths and Support Libraries

Determine key to matching operator using clustering

Optional

libcluster

Subscriber

TCP
Acast
Unicast

libfilter

Optional

AP

M

M

Store subscription in operator slice state

DCCP

State

C

Static component

O

Operator slice

lib

Library

C

StreamHub client

Sunday 6 July 14
Events Paths and Support Libraries

```
Determine key to matching operator using clustering

libcluster

S

DCCP

TCP

A

Any cast

Unicast

M

Store subscription in operator slice state

libfilter

Filter publication, return matching subscribers list

Publication + subscribers

DCCP

p

Publisher

TCP

A

Any cast

Broadcast

Unicast

M

Multicast

EP

DCCP

p

Subscriber

DCCP

TCP

...```

Optional

TCP

DCCP

AP

Unicast

EP

M

Static component

Operator slice

Library

State

StreamHub client

ICDCS 2014 - Etienne Rivière

Sunday 6 July 14
StreamHub interface

- Data Conversion and Connection Point(s) (DCCP)
  - Stateless component(s) running on server(s) with direct WAN connectivity
  - Maintain persistent TCP connections to/from publishers and subscribers
  - Convert between external/internal format
Use an Overlay of Brokers?

- Brokers organized in an overlay (mesh, tree); each broker performs all pub/sub operations
  - Storing subscriptions; matching, forwarding, notifying of publications
  - Complex maintenance of routing tables between brokers

- Assumptions on the filtering scheme for inter-broker communication and publication forwarding: containment & aggregation
  - Scalability/throughput depend on workload, notification delays may vary
Use an Overlay of Brokers?

- Brokers organized in an overlay (mesh, tree); each broker performs all pub/sub operations
  - Storing subscriptions; matching, forwarding, notifying of publications
  - Complex maintenance of routing tables between brokers

- Assumptions on the filtering scheme for inter-broker communication and publication forwarding: containment & aggregation
  - Scalability/throughput depend on workload, notification delays may vary