The World Cup of Event Processing

Big Data Research @ UofT, July 16 2014

Middleware Systems Research Group
University of Toronto

Kaiwen Zhang

Collaborators:
Hans-Arno Jacobsen
Kianoosh Mokhtarian
Tilmann Rabl
Mohammad Sadoghi
Reza Sherafat Kazemzadeh
Young Yoon

www.padres.msrg.utoronto.ca
What is event processing?

Event Source:
- Systems
- Business Processes
- Sensors

Event Processor

Events

Actions

Event Consumer
What is event processing?

Unit of datum:
- Semantic events
- Collected metrics
What is event processing?

Process data according to continuous queries (subscriptions)
What is event processing?

Output:
- Event notifications
- Aggregated reports
- Automated workflows
Big data challenge: Application performance monitoring
Big data challenge: Application performance monitoring
Big data challenge: Application performance monitoring
Big data challenge:
Application performance monitoring
Big data challenge: Application performance monitoring
# Data scale in APM systems

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Value</th>
<th>Min. Value</th>
<th>Max. Value</th>
<th>Data Points</th>
<th>Start Time (millis)</th>
<th>Stop Time (millis)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontends/ApplicationX: AverageResponseTime (ms)</td>
<td>3</td>
<td>2.0</td>
<td>4.0</td>
<td>2</td>
<td>1314121455000</td>
<td>1314121455000</td>
</tr>
</tbody>
</table>
# Data scale in APM systems

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Value</th>
<th>Min. Value</th>
<th>Max. Value</th>
<th>Data Points</th>
<th>Start Time</th>
<th>Stop Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontends</td>
<td>ApplicationX: AverageResponseTime (ms)</td>
<td>3</td>
<td>2.0</td>
<td>4.0</td>
<td>2</td>
<td>1314121455000</td>
</tr>
</tbody>
</table>

# Nodes: 100-10K
# Data scale in APM systems

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Value</th>
<th>Min. Value</th>
<th>Max. Value</th>
<th>Data Points</th>
<th>Start Time (millis)</th>
<th>Stop Time (millis)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontends</td>
<td>ApplicationX: AverageResponseTime (ms)</td>
<td>3</td>
<td>2.0</td>
<td>4.0</td>
<td>2</td>
<td>1314121455000</td>
</tr>
</tbody>
</table>

# Nodes: 100-10K

Metrics/node: up to 50K, average 10K
Data scale in APM systems

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Value</th>
<th>Min. Value</th>
<th>Max. Value</th>
<th>Data Points</th>
<th>Start Time (millis)</th>
<th>Stop Time (millis)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontends</td>
<td>ApplicationX: AverageResponseTime (ms)</td>
<td>3</td>
<td>2.0</td>
<td>4.0</td>
<td>2</td>
<td>1314121455000</td>
</tr>
</tbody>
</table>

# Nodes: 100-10K

Metrics/node: up to 50K, average 10K

Reporting period: average 10 seconds
## Data scale in APM systems

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Value</th>
<th>Min. Value</th>
<th>Max. Value</th>
<th>Data Points</th>
<th>Start Time (millis)</th>
<th>Stop Time (millis)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontends</td>
<td>ApplicationX: AverageResponseTime (ms)</td>
<td>3</td>
<td>2.0</td>
<td>4.0</td>
<td>2</td>
<td>1314121455000</td>
</tr>
</tbody>
</table>

- # Nodes: 100-10K
- Metrics/node: up to 50K, average 10K
- Reporting period: average 10 seconds
- Event rate: 1M/sec
## Data scale in APM systems

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Value</th>
<th>Min. Value</th>
<th>Max. Value</th>
<th>Data Points</th>
<th>Start Time (millis)</th>
<th>Stop Time (millis)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontends</td>
<td>ApplicationX: AverageResponseTime (ms)</td>
<td>3</td>
<td>2.0</td>
<td>4.0</td>
<td>2</td>
<td>1314121455000</td>
</tr>
</tbody>
</table>

- **# Nodes:** 100-10K
- **Metrics/node:** up to 50K, average 10K
- **Reporting period:** average 10 seconds
- **Event rate:** 1M/sec
- **Data size:** 100B/event
Data scale in APM systems

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Value</th>
<th>Min. Value</th>
<th>Max. Value</th>
<th>Data Points</th>
<th>Start Time (millis)</th>
<th>Stop Time (millis)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontends</td>
<td>ApplicationX: AverageResponseTime (ms)</td>
<td>3</td>
<td>2.0</td>
<td>4.0</td>
<td>2</td>
<td>1314121455000</td>
</tr>
</tbody>
</table>

- # Nodes: 100-10K
- Metrics/node: up to 50K, average 10K
- Reporting period: average 10 seconds
- Event rate: 1M/sec
- Data size: 100B/event
- Raw data: 100MB/sec, 355GB/h, 2.8 PB/y
Soccer monitoring

DEBS 2013 Grand Challenge
Dataset provided by Fraunhofer (54M events)
Four challenge queries
(Video)
Monitoring architecture
Monitoring architecture

Data collection and dispatching

distribution stage

CEP engine

socket comm. layer

event processing stage
Monitoring architecture

Event processing stage

data collection and dispatching stage

distribution stage

Event processing stage

CEP engine

socket comm. layer
Monitoring architecture

Event processing stage

data collection and dispatching stage

distribution stage

event processing stage
Monitoring architecture

log file → network data dispatcher → Distribution stage

data collection and dispatching stage → event processing stage

distribution stage
GUI client (video)

http://msrg.org/datasets/blue-bay
World cup of event processing
World cup of event processing

Esper:
CEP Engine + Language
World cup of event processing

Esper: CEP Engine + Language

Storm: MapReduce-like
World cup of event processing

StreamIt: CEP Language+Compiler

Storm: MapReduce-like
World cup of event processing

StreamIt: CEP Language+Compiler

Stream: DSMS
World cup of event processing

BlueBay: Custom engine in C++
World cup of event processing

Esper

BlueBay

Stanford Streamdata Manager

StreamIt
World cup of event processing

Library support for common languages such as Java
Query #2 (Ball possession)

```sql
insert into b_possession_percent
select *
    , sum(b_ts - prev(b_ts, 1)) as time_total
    , sum((b_ts - prev(b_ts, 1))
        * msrg.GameSetting.equalStr(owner,prev(owner,1),'teamA'))
        as time_teamA
    , sum((b_ts - prev(b_ts, 1))
        * msrg.GameSetting.equalStr(owner,prev(owner,1),'teamB'))
        as time_teamB
from b_possession.win:time(10 seconds)
```
Esper

Query #2 (Ball possession)

```sql
insert into b_possession_percent
select *
, sum(b_ts - prev(b_ts, 1)) as time_total,
  sum((b_ts - prev(b_ts, 1))
  * msrg.GameSetting.equalStr(owner, prev(owner, 1), 'teamA'))
  as time_teamA,
  sum((b_ts - prev(b_ts, 1))
  * msrg.GameSetting.equalStr(owner, prev(owner, 1), 'teamB'))
  as time_teamB
from b_possession.win:time(10 seconds)
```

Use of Esper stream primitives
To process the stream
Query #2 (Ball possession)

```sql
insert into b_possession_percent
select *
, sum(b_ts - prev(b_ts, 1)) as time_total,
  sum((b_ts - prev(b_ts, 1))
    * msrg.GameSetting.equalStr(owner,prev(owner,1),'teamA'))
  as time_teamA,
  sum((b_ts - prev(b_ts, 1))
    * msrg.GameSetting.equalStr(owner,prev(owner,1),'teamB'))
  as time_teamB
from b_possession.win:time(10 seconds)
```

Use of user-defined helper functions
Query #2 (Ball possession)

```sql
insert into b_possession_percent
select *
    , sum(b_ts - prev(b_ts, 1)) as time_total
    , sum((b_ts - prev(b_ts, 1)) * msrg.GameSetting.equalStr(owner, prev(owner, 1), 'teamA')) as time_teamA
    , sum((b_ts - prev(b_ts, 1)) * msrg.GameSetting.equalStr(owner, prev(owner, 1), 'teamB')) as time_teamB
from b_possession.win:time(10 seconds)
```

Use of Esper window semantics
To extract stream at correct granularity
Query #3 (Heatmap)
Query #3 (Heatmap)

Sensor Events → Player ID → Player Position → Player Heatmap → Output

Split the stream (Map)
Query #3 (Heatmap)

- Sensor Events
- Player ID
- Player Position
- Player Heatmap
- Output

Split the stream (Map)

Project right attributes
Query #3 (Heatmap)

Split the stream (Map)

Project right attributes

User-defined functions

Sensor Events -> Player ID -> Player Position -> Player Heatmap -> Output
Query #3 (Heatmap)

Sensor Events → Player ID → Player Position → Player Heatmap → Output

- Split the stream (Map)
- Project right attributes
- Merge output (Reduce)

User-defined functions
BlueBay origins
BlueBay origins
BlueBay architecture

Player Tracker
Active-Ball Detector
Ball Tracker

Query 1 Handler
Intensities
Stream Window

Query 4 Handler
None
Wait for trajectory
Goal shot

Query 2 Handler
Team 1
Stream Window
ArrayMap<Player, Time>
LastHitterPlayer
LastHitTimestamp
...

Query 3 Handler
Player 1
Resolution 1
Stream Window

Player 2
Resolution 2
Stream Window

Player 3
BlueBay architecture

Stream components per object w/ noise reduction

Player Tracker
Active-Ball Detector
Ball Tracker

Query 1 Handler
Intensities
Stream Window

Query 4 Handler
None
Wait for trajectory
Goal shot

Query 2 Handler
Team 1
Stream Window
Team 2
Stream Window

ArrayMap<Player, Time>
LastHitterPlayer
LastHitTimestamp
...

Query 3 Handler
Player 3
Player 2
Player 1
Resolution 1
Resolution 2
Stream Window
Stream Window
BlueBay architecture

Modular design allows for intra- and inter-queries parallelization

Query 1 Handler
- Intensities
  - Stream Window

Query 2 Handler
- Team 1
  - Stream Window
- Team 2
  - Stream Window
- ArrayMap<Player, Time>
- LastHitterPlayer
- LastHitTimestamp
...

Query 3 Handler
- Player 1
  - Resolution 1
    - Stream Window
  - Resolution 2
    - Stream Window

Query 4 Handler
- None
  - Wait for trajectory
  - Goal shot
BlueBay architecture

Circular list of timestamp range buckets O(1) time and memory
BlueBay architecture
# Experimental comparison

<table>
<thead>
<tr>
<th></th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlueBay</td>
<td>141x</td>
<td>165x</td>
<td>30x</td>
<td>187x</td>
</tr>
<tr>
<td>Esper</td>
<td>7.5x</td>
<td>2.4x</td>
<td>6.3x</td>
<td>2.3x</td>
</tr>
<tr>
<td>Storm</td>
<td>9.7x</td>
<td>8.6x</td>
<td>9.8x</td>
<td>8.6x</td>
</tr>
</tbody>
</table>
## Experimental comparison

<table>
<thead>
<tr>
<th></th>
<th>Q3</th>
<th>Q4</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlueBay</td>
<td>141x</td>
<td>165x</td>
</tr>
<tr>
<td>Esper</td>
<td>7.5x</td>
<td>2.4x</td>
</tr>
<tr>
<td>Storm</td>
<td>9.7x</td>
<td>8.6x</td>
</tr>
</tbody>
</table>

*Running in single-threaded mode*
Impact of multi-threading

- Non-threaded
- Threaded ($q = 4$)
- Threaded ($q = 1k$)
- Threaded ($q = 16k$)

Events per second vs Time (minutes)

- x1000 events per second
- 0 to 2.5 minutes
Impact of multi-threading

Queue size to synchronize the queries

- Threaded \( (q = 4) \)
- Threaded \( (q = 1k) \)
- Threaded \( (q = 16k) \)
Impact of multi-threading

Throughput bounded by slowest query
Impact of multi-threading

- Non-threaded
- Threaded \( (q = 4) \)
- Threaded \( (q = 1k) \)
- Threaded \( (q = 16k) \)

Missing ball data in workload
Impact of multi-threading

Highest sustained average 790k e/s

Threaded (q = 4)

Threaded (q = 1k)

Threaded (q = 16k)
Impact of multi-threading

BlueBay: 60x speedup!
Conclusion
Conclusion
Conclusion

Continuous evaluation of standing queries over potentially infinite streams of data
Conclusion

Wide range of applications:
• Business process management
• Application performance monitoring
• Sensor networks
Conclusion

Big Data scale: How to handle 1M events/s?
Conclusion
Conclusion

Three finalist: Esper, Storm, BlueBay
Conclusion

BlueBay performance:
790k evts/s, 60x speedup
Oslo, Norway (June 29 – July 3, 2015)
Thank you!

Oslo, Norway (June 29 – July 3, 2015)
References


Advertisement-based Publish/Subscribe model
Advertisement-based
Publish/Subscribe model

broker

Advertisement path
Subscription path
Publication path
Advertisement-based Publish/Subscribe model

- Advertisement path
- Subscription path
- Publication path

- publisher
- broker

name = 'IBM'
price is present
Advertisement-based Publish/Subscribe model

- Advertisement path
- Subscription path
- Publication path

publisher

broker

name = ‘IBM’
price is present
Advertisement-based Publish/Subscribe model

**Advertisement path**

**Subscription path**

**Publication path**

<table>
<thead>
<tr>
<th>Publisher</th>
<th>Broker</th>
<th>Subscriber</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name = 'IBM'</code></td>
<td><code>name = 'IBM'</code></td>
<td><code>name = 'IBM'</code></td>
</tr>
<tr>
<td><code>price is present</code></td>
<td><code>price &gt; $20</code></td>
<td></td>
</tr>
</tbody>
</table>
Advertisement-based Publish/Subscribe model

Publisher

name = 'IBM'
price is present

Subscription path
Publication path

Subscriber

name = 'IBM'
price > $20

broker

Advertisement path
Advertisement-based Publish/Subscribe model

- **Publisher**: name = ‘IBM’, price is present
- **Subscription path**: three subscribers
  - **Subscriber 1**: name = ‘IBM’, price > $20
  - **Subscriber 2**: name = ‘IBM’, price > $30
  - **Subscriber 3**: name = ‘IBM’, price is present
- **Publication path**: three subscribers
  - **Subscriber 1**: name = ‘IBM’, price > $20
  - **Subscriber 2**: name = ‘IBM’, price > $30
  - **Subscriber 3**: name = ‘IBM’, price is present

**Advertisement path**

- **Broker**
Advertisement-based Publish/Subscribe model

- Advertisement path
- Subscription path
- Publication path

**Publisher**
- name = 'IBM'
- price is present

**Subscribers**
- name = 'IBM'
- price > $20
- price > $30
Advertisement-based Publish/Subscribe model

- Advertisement path
- Subscription path
- Publication path

Publisher:
- name = `IBM`
- price = $40

Broker:
- match & forward

Subscriber 1:
- name = `IBM`
- price > $20

Subscriber 2:
- name = `IBM`
- price > $30
Throughput-latency tradeoff

![Graph showing throughput-latency tradeoff](image)

- **Average**
- **99-percentile**

**Axes:**
- X-axis: Throughput (x1000 events per second)
- Y-axis: Delay (ms)

**Legend:**
- Blue square: Average
- Red circle: 99-percentile
Throughput-latency tradeoff

Time required to fully process one input event for all queries

Delay (ms)

Throughput (x1000 events per second)

Average

99-percentile

Combined throughput of output events for all queries
Throughput-latency tradeoff

Less throughput means less queue size

Average

99-percentile
Throughput-latency tradeoff

Less queue size blocks faster threads.
Give full CPU usage to slower threads, clearing older events.

Less throughput means less queue size.
Throughput-latency tradeoff

Large queue size allows fast threads to process and emit events quickly and ahead of slower threads.