StreamCloud: an Elastic Parallel-Distributed Stream Processing Engine

Ph.D. Student
Vincenzo Massimiliano Gulisano

Director: Ricardo Jiménez Peris
Co-director: Patrick Valduriez

DISTRIBUTED SYSTEMS LABORATORY

December 20, 2012
StreamCloud in a Nutshell

• Sample data streaming application (fraud detection)
• Pioneer Stream Processing Engines (SPEs)
• Contributions
Motivation - Fraud detection
Motivation - Fraud detection
Motivation - Fraud detection
Motivation - Fraud detection
Background - Pioneer SPEs
Background - Pioneer SPEs

Centralized SPE

100% CPU
Background - Pioneer SPEs

Distributed SPE
Background - Pioneer SPEs
Background - Pioneer SPEs
Background - Pioneer SPEs

Distributed SPE

100% CPU
Contributions - StreamCloud
Contributions - StreamCloud
Contributions - StreamCloud

1 Parallelization
Contributions - StreamCloud
Contributions - StreamCloud
Contributions - StreamCloud

2 Elasticity
Contributions - StreamCloud
Contributions - StreamCloud
Contributions - StreamCloud
Contributions - StreamCloud

3 Fault Tolerance
Contributions - StreamCloud
Contributions - StreamCloud
Contributions - StreamCloud

4 Integrated Development Environment
Agenda

• Introduction
  — Motivation
  — System Model
• Parallelization
• Elasticity
• Fault tolerance
• Integrated Development Environment
• Conclusions
Motivation

• Financial applications, sensor networks monitoring, ... require
  – Continuous processing of data streams
  – Real Time fashion

• Store and process is not feasible
  – high-speed networks, nanoseconds to handle a packet
  – ISP router: gigabytes of headers every hour,…

• Data Streaming:
  – In memory
  – Bounded resources
  – Efficient one-pass analysis
System Model

- Data Stream: unbounded sequence of tuples
  - Example: Call Description Record (CDR)

<table>
<thead>
<tr>
<th>Field</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caller</td>
<td>text</td>
</tr>
<tr>
<td>Callee</td>
<td>text</td>
</tr>
<tr>
<td>Time (secs)</td>
<td>int</td>
</tr>
<tr>
<td>Price (€)</td>
<td>double</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>8:00</th>
<th>3</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>D</td>
<td>8:20</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>E</td>
<td>8:35</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

A B 8:00 3 C D 8:20 7 A E 8:35 6
System Model

• Operators:

<table>
<thead>
<tr>
<th>OP</th>
<th>Stateless</th>
<th>OP</th>
<th>Stateful</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 input tuple</td>
<td></td>
<td>1+ input tuple(s)</td>
</tr>
<tr>
<td></td>
<td>1 output tuple</td>
<td></td>
<td>1 output tuple</td>
</tr>
</tbody>
</table>
System Model

• Operators:

<table>
<thead>
<tr>
<th>Stateless</th>
<th>Stateful</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP</td>
<td>OP</td>
</tr>
<tr>
<td>1 input tuple</td>
<td>1+ input tuple(s)</td>
</tr>
<tr>
<td>1 output tuple</td>
<td>1 output tuple</td>
</tr>
</tbody>
</table>

• Continuous Query: graph operators/streams

  Map → Filter → Agg

  - Map: Convert € → $
  - Filter: Only > 10$
  - Agg: Count calls made by each Caller number
System Model

- Infinite sequence of tuples / bounded memory → windows
- Example: 1 hour windows

\[ [8:00,9:00) \]
\[ [8:20,9:20) \]
\[ [8:40,9:40) \]
System Model

• Infinite sequence of tuples / bounded memory
  → windows

• Example: 1 hour windows

Counter: 1
System Model

- Infinite sequence of tuples / bounded memory
  → windows
- Example: 1 hour windows
System Model

- Infinite sequence of tuples / bounded memory → windows
- Example: 1 hour windows

Counter: 3

[8:00,9:00)
System Model

• Infinite sequence of tuples / bounded memory → windows

• Example: 1 hour windows

Counter: 4
System Model

• Infinite sequence of tuples / bounded memory
  → windows

• Example: 1 hour windows

Counter: 4

Output: 4
System Model

• Infinite sequence of tuples / bounded memory
  \( \rightarrow \) windows

• Example: 1 hour windows

Counter: 3
Agenda

• Introduction
  — Motivation
  — System Model
• Parallelization
• Elasticity
• Fault tolerance
• Integrated Development Environment
• Conclusions
StreamCloud - Parallelization

• Building blocks:
  – Parallelization of data streaming operators
  – Parallelization and Distribution strategy
StreamCloud - Parallelization

• General approach

\[ \text{OP}_A \rightarrow \text{OP}_B \]
StreamCloud - Parallelization

• General approach

LB: Load Balancer
IM: Input Merger
StreamCloud - Parallelization

• General approach

Subcluster A

IM → OP_A → LB

Node 1

... 

IM → OP_A → LB

Node m

OP_A → OP_B

LB: Load Balancer
IM: Input Merger
StreamCloud - Parallelization

• General approach

Subcluster A

Subcluster B

IM → OP_A → LB
Node 1

IM → OP_A → LB
Node m

IM → OP_A → LB
Node 1

IM → OP_B → LB
Node n

OP_A → OP_B

LB: Load Balancer
IM: Input Merger
StreamCloud - Parallelization

• General approach
StreamCloud - Parallelization

• Stateful operators: Semantic awareness
  – Aggregate: count within last hour, group-by caller number

Previous Subcluster

LB

LB

IM

IM

IM

Agg₁

Agg₂

Agg₃

Caller A
StreamCloud - Parallelization

• **Stateful operators: Semantic awareness**
  
  – **Aggregate**: count within last hour, group-by caller number
StreamCloud - Parallelization

• Depending on the stateful operator semantic:
  – Partition input stream into **buckets**
  – Each bucket is processed by 1 node

• # buckets >> # nodes
StreamCloud - Parallelization

- Depending on the stateful operator semantic:
  - Partition input stream into **buckets**
  - Each bucket is processed by 1 node
- # buckets >> # nodes

![Diagram of keys and aggregates]

**Keys domain**

- A
- B
- C
- D
- E
- F

**Aggregates**

- Agg₁
- Agg₂
- Agg₃
StreamCloud - Parallelization

- Depending on the stateful operator semantic:
  - Partition input stream into buckets
  - Each bucket is processed by 1 node

- # buckets >> # nodes
StreamCloud - Parallelization

- Example:

\[ \text{OP}_1 \rightarrow \text{OP}_2 \rightarrow \text{OP}_3 \rightarrow \text{OP}_4 \rightarrow \text{OP}_5 \rightarrow \text{OP}_6 \]
StreamCloud - Parallelization

• Example:

\[ \text{OP}_1 \rightarrow \text{OP}_2 \rightarrow \text{OP}_3 \rightarrow \text{OP}_4 \rightarrow \text{OP}_5 \rightarrow \text{OP}_6 \]

– Cluster composed by 90 nodes
StreamCloud - Parallelization

• Example:

- Cluster composed by 90 nodes

How to parallelize/distribute the query to maximize throughput?
StreamCloud - Parallelization

- Parallelization Strategies

  - Full query at all nodes
  - 1+ operators per node
  - 1 operator per node
StreamCloud - Parallelization

![Diagram showing parallelization in StreamCloud]

- OP1 → OP2 → OP3 → OP4 → OP5 → OP6
- Match Caller
- Match Callee

Full query at all nodes
StreamCloud - Parallelization

Introduction Parallelization Elasticity Fault Tolerance IDE Conclusions

Full query at all nodes
StreamCloud - Parallelization

1 operator per node
StreamCloud - Parallelization

1 operator per node
StreamCloud - Parallelization

Introduction

Parallelization

Elasticity

Fault Tolerance

IDE

Conclusions

1+ operators per node

Node 1

Node 31

Node 61

Node 30

Node 60

Node 90
StreamCloud - Parallelization

1+ operators per node
StreamCloud - Parallelization

- Scalability of data streaming operators
- Aggregate operator:
  Compute average call duration, number of calls for each phone number
- Setup:
  - 1 / 20 / 40 nodes
StreamCloud - Parallelization

- Aggregate scale out evaluation

![Graph showing throughput vs load for different node counts](image)
StreamCloud - Parallelization

- Aggregate scale out evaluation

![Graph showing throughput vs load for different node counts](image)
StreamCloud - Parallelization

- Aggregate scale out evaluation

![Graph showing throughput vs load for different node counts](image)

- For 1 Node: ~12000 t/s
- For 20 Nodes: ~210000 t/s
- For 40 Nodes: ~210000 t/s

Relative increase:
- 20 Nodes vs 1 Node: x19
- 40 Nodes vs 1 Node: x21
StreamCloud - Parallelization

- Aggregate scale out evaluation

![Graph showing throughput vs load](image)
Agenda

• Introduction
  – Motivation
  – System Model
• Parallelization
• Elasticity
• Fault tolerance
• Integrated Development Environment
• Conclusions
StreamCloud - Elasticity

• Building blocks:
  – Elasticity rules
  – Dynamic Load Balancing combined with provisioning / decommissioning
  – Load-aware provisioning / decommissioning
StreamCloud - Elasticity

Elasticity and load balancing actions on per-subcluster basis
StreamCloud - Elasticity

Elasticity and load balancing actions on per-subcluster basis

- Upper utilization threshold
- Target utilization threshold
- Lower utilization threshold

CPU consumption

Introduction Parallelization Elasticity Fault Tolerance IDE Conclusions
StreamCloud - Elasticity

• Transfer state between nodes
• Load balancing algorithm

• Minimum parallelization unit $\rightarrow$ bucket
  – Transfer buckets between instances
StreamCloud - Elasticity

• State transfer challenging for stateful operators
StreamCloud - Elasticity

- State transfer challenging for stateful operators
StreamCloud - Elasticity

• State transfer challenging for stateful operators
StreamCloud - Elasticity

- Window Recreation Protocol

![Diagram showing tuples and nodes A and B over time.](image)
StreamCloud - Elasticity

- Window Recreation Protocol

![Diagram](image)

- Tuples referring to caller A
- Time

A

B
StreamCloud - Elasticity

• Window Recreation Protocol

- No communication between nodes
- Completion time proportional to window size
StreamCloud - Elasticity

• State Recreation Protocol

Tuples referring to caller A
StreamCloud - Elasticity

- State Recreation Protocol

![Diagram showing state recreation protocol]

Tuples referring to caller A

Copy to B

time
StreamCloud - Elasticity

- State Recreation Protocol

+ Minimizes completion time
- Communication between nodes
StreamCloud - Elasticity

• Load balancing algorithm:
• Finding the right assignment of buckets to machines → bin packing problem (NP hard)
• Should reduce the number of state transfers (cannot reallocate all buckets)
StreamCloud - Elasticity

- **Greedy algorithm:**
  - Balance load \(\rightarrow\) Minimized CPU standard deviation

1. Sort nodes on CPU utilization
2. Transfer heaviest bucket of first node to last node
3. Standard Deviation decrease > threshold? 

Stop

Introduction Parallelization Elasticity Fault Tolerance IDE Conclusions
StreamCloud - Elasticity

• Load-aware provisioning evaluation
• Query: High Mobility (cellular telephony fraud detection)
StreamCloud - Elasticity

• Load-aware provisioning evaluation
• Query: High Mobility (cellular telephony fraud detection)

Extract time and position for each pair of consecutive CDR from the same Caller
StreamCloud - Elasticity

- Load-aware provisioning evaluation
- Query: High Mobility (cellular telephony fraud detection)

Extract time and position for each pair of consecutive CDR from the same Caller
StreamCloud - Elasticity

- Load-aware provisioning

![Graph showing load-aware self provisioning](image_url)
StreamCloud - Elasticity

- Load-aware provisioning

![Graph showing load-aware self provisioning from 1 to 26 with throughput, CPU usage, and cluster size over time.](image)
Agenda

• Introduction
  — Motivation
  — System Model
• Parallelization
• Elasticity
• Fault tolerance
• Integrated Development Environment
• Conclusions
StreamCloud - Fault Tolerance

• Studied in the context of distributed SPEs

• Existing techniques:
  – Active standby
  – Passive standby
  – Upstream backup
Fault Tolerance – State of the Art

• Active standby
Fault Tolerance – State of the Art

• Passive standby
Fault Tolerance – State of the Art

• Upstream Backup
StreamCloud - Fault Tolerance

• Low-cost redundancy
StreamCloud - Fault Tolerance

• Low-cost redundancy
  ➔ Move redundancy at the storage level (cheaper)
StreamCloud - Fault Tolerance

• Low-cost redundancy
  ➔ Move redundancy at the storage level (cheaper)

• Precise Recovery
StreamCloud - Fault Tolerance
StreamCloud - Fault Tolerance
StreamCloud - Fault Tolerance
StreamCloud - Fault Tolerance
StreamCloud - Fault Tolerance
StreamCloud - Fault Tolerance

Maintain information about the last processed tuples
StreamCloud - Fault Tolerance

• Low-cost redundancy
  ➔ Move redundancy at the storage level (cheaper)

• Precise Recovery
  ➔ Maintain information about last processed tuples
StreamCloud - Fault Tolerance

• Low-cost redundancy
  → Move redundancy at the storage level (cheaper)
• Precise Recovery
  → Maintain information about last processed tuples
• Decouple State Maintenance – Topology
StreamCloud - Fault Tolerance

- Low-cost redundancy
  - Move redundancy at the storage level (cheaper)
- Precise Recovery
  - Maintain information about last processed tuples
- Decouple State Maintenance – Topology
  - System is not STATIC → No point in check-pointing nodes
StreamCloud - Fault Tolerance

• Low-cost redundancy
  → Move redundancy at the storage level (cheaper)

• Precise Recovery
  → Maintain information about last processed tuples

• Decouple State Maintenance – Topology
  – System is not STATIC → No point in check-pointing nodes
  – Maintain past tuples of buckets
StreamCloud - Fault Tolerance
StreamCloud - Fault Tolerance
StreamCloud - Fault Tolerance
StreamCloud - Fault Tolerance

Introduction  Parallelization  Elasticity  Fault Tolerance  IDE  Conclusions
StreamCloud - Fault Tolerance

• Evaluation – Runtime overhead
• Excerpt of the Linear Road benchmark query
  – Operators used to detect accidents
  – 1 aggregate + 1 filter
• 10 nodes
• Input load ~30000 tuples / second
StreamCloud - Fault Tolerance

![Graph showing latency over time with a blue line for FT and a gray line for No-FT. The x-axis represents time in seconds (0 to 600), and the y-axis represents latency in milliseconds (55 to 60).]
StreamCloud - Fault Tolerance

- ~3ms
- ~1ms <2%

No-FT vs FT comparison graph showing latency over time with improved fault tolerance.
Agenda

• Introduction
  – Motivation
  – System Model
• Parallelization
• Elasticity
• Fault tolerance
• Integrated Development Environment
• Conclusions
StreamCloud - IDE

• Ease the user interaction
  – Which is the min information to provide to run a data streaming application?
    • Query operators
    • Available nodes
StreamCloud - IDE

Visual Query Composer

[Diagram of Visual Query Composer with various operators and arrows indicating input and output]
StreamCloud - IDE

Visual Query Composer
StreamCloud - IDE

Visual Query Composer
StreamCloud - IDE

• Ease the user interaction
  – Which is the min information to provide to run a data streaming application?
    • Query operators
    • Available nodes

• Tools
  – Parallel Compiler
  – Deployer
  – Distributed Load injectors
  – Monitoring of queries
StreamCloud - IDE

• Monitoring of queries

• For each query operator, provides the following statistics:
  • Input Stream Rate
  • Output Stream Rate
  • Cost
  • Queue Length
  • CPU
  • # of Nodes
StreamCloud - IDE

- Monitoring of queries
Agenda

- Introduction
  - Motivation
  - System Model
- Parallelization
- Elasticity
- Fault tolerance
- Integrated Development Environment
- Conclusions
Conclusions

• StreamCloud
  – Parallelization of data streaming operators
  – Load-aware Self Provisioning – Decommissioning and dynamic load balancing
  – Fault Tolerance
  – IDE to ease user interaction
Conclusions

• Selected publications:

• International Journals
  – StreamCloud: An Elastic and Scalable Data Streaming System.  
    V. Gulisano, R. Jimenez-Peris, M. Patiño-Martinez, C. Soriente, P. Valduriez - IEEE Transactions on Parallel and Distributed Systems (TPDS)

• International Conferences and other publications:
    M.Callau, V. Gulisano, Z. Fu, R. Jimenez-Peris, M. Papatriantafilou, M. Patiño-Martinez 
    28th Annual ACM Symposium on Applied Computing (SAC) 2013
  – StreamCloud: A Large Scale Data Streaming System 
    V. Gulisano, R. Jimenez-Peris, M. Patiño-Martinez, P. Valduriez 
    30th Int. Conf. on Distributed Computing Systems (ICDCS) 2010
  – A New Class of Services for Cloud Computing: Real-Time Services over Massive and Continuous Data Flows. 
    Ricardo Jimenez-Peris, M. Patiño-Martinez, V. Gulisano 
    Cloud Futures 2010 Workshop

• Book Chapters
  – Complex Event Processing Based SIEM 
    V. Gulisano, R. Jimenez-Peris, M. Patiño-Martinez, C. Soriente, V. Vianello 
    Advances in Security Information Management: Perceptions and Outcomes. 2011
StreamCloud: an Elastic Parallel-Distributed Stream Processing Engine

Ph.D. Student
Vincenzo Massimiliano Gulisano

Director: Ricardo Jiménez Peris
Co-director: Patrick Valduriez

Lsd
DISTRIBUTED
SYSTEMS
LABORATORY

December 20, 2012