Distributed Strategies for Elastic Data Stream Processing in the Fog

Valeria Cardellini, Francesco Lo Presti, Matteo Nardelli, Gabriele Russo Russo

University of Rome Tor Vergata, Italy
Data Stream Processing

**Data Stream Processing (DSP) applications:**
- processing of *data streams* generated by distributed sources
- extract information in a (near) *real-time* manner
- in memory processing

Twitter sentiment analysis based on DSP Framework Apache Storm ‘til 2014
Well suited for Smart City Applications
  - To increase scalability and availability, reduce latency, network traffic, etc...

Exploit fog and near-edge computation
(distributed cloud and Fog computing)
Old and New Challenges

Distributed Environment

• Geographic distribution, network latencies are not-negligible
• Data cannot be quickly moved among computing nodes

DSP Applications are long running

• Subject to varying load, network variability

Reconfigure the application deployment

• has a non negligible cost!
• can negatively affect application performance in the short term
  – Application freezing times, especially for stateful operators
State of the art

Centralized approaches:
• most of the proposed approaches designed for clusters
• Architecture (and control algorithms) do not scale well in a distributed environment

Decentralized approaches:
• several proposal
• their inherent lack of coordination might result in frequent reconfigurations

MAPE (Monitor, Analyze, Plan and Execute)
Decentralized MAPE

- Many Patterns, each with pro and cons

Goals

• Design a **hierarchical distributed approach** to the autonomous control of DSP applications

• Support **run-time adaptation**
  – **Elasticity**
    automatically scale in/out the number of operator instances
  – **Stateful Migration**
    relocate operators without compromising application integrity

• Design a **control policy**

• Integration of our solution in Storm
Hierarchical MAPEs in Storm

• New components in Apache Storm to realize a Hierarchical MAPE pattern

• **Operator Manager vs Application Manager**
  – Concerns and time scale separation
Hierarchical MAPEs in Storm

Operator Manager

- Monitors operator and local resources
  - e.g., Thread CPU utilization,
- Determines whether a Migration and/or Scale operation is needed
- Executes the reconfiguration
  - If gets the permission to

Application Manager

- Monitors Application Performance
  - SLA enforcement
- Coordinates operator reconfigurations
  - Grants permission to enact reconfigurations
  - Controls reconfiguration frequencies

General Framework for Distributed Optimization
Simple Distributed Heuristic: Operator Manager

• issues reconfiguration plans:
  
  **action, gain, cost**

• action: *migrates* an operator replica
  – threshold based policy on CPU utilization
  – new location: probabilistic selection from the neighborhood
  – cost: estimated stateful migration time

• action: *operator scaling*
  – Threshold based: $U_{th}$
  1. **scale out**: replicate replica $i$ if $U_i > U_{th}$
  2. **scale in**: remove one of the $n$ replicas if removing it does not overload
     the others -> $\sum U_i / (n-1) < 0.75 U_{th}$
     – cost: estimated time to relocate the operator state (if any)

• gain function: *scale-out > migration > scale-in*
Token-based policy

- Considers time divided in control intervals
- Generates reconfiguration tokens based on application performance
- Grants as many reconfigurations as available tokens
  - Prioritizing by gain to cost ratio
Evaluation

Infrastructure
• 5 worker nodes + 1 host for Nimbus and ZooKeeper
• each node Intel Xeon 8 cores@2Ghz, 16 GB RAM

Application
• DEBS 2015 Grand Challenge: **top10 frequent routes** NYC taxis in the last 30 min
• Requires: max Response Time $R_{\text{max}} = 200$ ms

Policy parameters
• Operator Manager policy: thresholds on utilization to 70%
• Application Manager policy: token bucket capacity = 1 token
Evaluation

Application Manager policy: grants all reconfiguration requests

Source data rate (tuples/s)

Response time (ms)

Total number of replicas

<table>
<thead>
<tr>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1000</td>
</tr>
<tr>
<td>2000</td>
</tr>
<tr>
<td>3000</td>
</tr>
<tr>
<td>4000</td>
</tr>
<tr>
<td>5000</td>
</tr>
</tbody>
</table>

- Median of response time: 130.6 ms
- Up and running: 93.7% of time
Evaluation

Application Manager policy: 1 token/min if response time > 50% R_{\text{max}}

- Source data rate (tuples/s)
- Response time (ms)
- Total number of replicas

- Graph showing source data rate with peaks and troughs indicating busy and idle periods.
- Graph showing response time with median value 117.0 ms.
- Graph showing total number of replicas with up and running 93.4% of the time.

- Diagram indicates scaling and migration actions at specific time points.
Application Manager policy: 1 token/min if response time > 75% $R_{\text{max}}$

- Source data rate (tuples/s)
- Response time (ms)
- Total number of replicas

Median of response time: 80.4 ms

Up and running: 98.3% of time
Conclusions

• We designed a hierarchical distributed architecture for the autonomous control of DSP applications

• We developed a simple control policy

• We integrated our solution in Storm

• We evaluated the effectiveness of our solution

Current Works

• Use MDP, Reinforcement Learning to design Control Policy
  – Team Markov Games
  – Multiple Agent Reinforcement Learning