Data Stream Processing: The Placement Problem
A.A. 2016/17

Matteo Nardelli

Laurea Magistrale in Ingegneria Informatica - II anno
Data Stream Processing

• Continuous processing of unbounded data streams generated by multiple, distributed sources, to extract valuable information in a timely and reliable manner

A new distributed environment

• Fog + Cloud computing: allows to increase scalability and availability, reduce latency, network traffic, and power consumption
Old and New Challenges

• Network latencies are significant
  – e.g., geo-distributed resources
• Computing and networking resources are heterogeneous
  – e.g., capacity limits, business constraints
• Computing/network resources not always available
• Data cannot be quickly moved around the network

Peculiarities of DSP Applications:
• computational requirements not known a-priori
• can change continuously
• load is imposed for long provisioning times

→ Need to adapt to internal and external changes
Operator Placement Problem

- To determine the (distributed) computing nodes that should host and execute each application operator, with the goal of optimizing the application QoS
Different Approaches

• Operator placement: **NP-hard** problem
• Several placement policies in literature (mainly heuristics) that address such problem but:
  – Different **assumptions** (system model, application topology, QoS attributes and metrics, …)
  – Different **objectives**
  – Not easily comparable

Three main approaches:
• Mathematical programming
  – formalization of the placement problem
  – this solution does not scale well, but provides useful insights
• Heuristics
• Graph theoretic
  – graph partitioning algorithms to schedule the DAG on a homogeneous cluster
Different Approaches

• Centralized solutions
  – Access to the entire resource and network state, application state, workload information (global view)
  **Pro:** Capable of determining optimal global solutions
  **Cons:** Scalability

• Decentralized solutions
  – Take decision based on local information
  **Pro:** Scalability, better suited for runtime adaptation
  **Cons:** Optimality is not guaranteed
ODP: Optimal DSP Placement

• We propose ODP
  – Centralized policy for optimal placement of DSP applications
  – Formulated as Integer Linear Programming (ILP) problem

• Our goals:
  – To compute the optimal placement (of course!)
  – To provide a unified general formulation of the placement problem for DSP applications (but not only!)
  – To consider multiple QoS attributes of applications and resources
  – To provide a benchmark for heuristics

Operators

• $C_i$ required computing resources
• $R_i$ execution time per data unit

Data streams

• $\lambda_{i,j}$ data rate from operator $i$ to $j$
Computing and network resources

**Computing resources**
- \( C_u \) amount of resources
- \( S_u \) processing speed
- \( A_u \) resource availability

**Logical) Network links**
- \( d_{u,v} \) network delay from \( u \) to \( v \)
- \( B_{u,v} \) bandwidth from \( u \) to \( v \)
- \( A_{u,v} \) link availability
ODP: Model

Decision variables

- Determine where to map operators and streams

\[ \begin{align*}
    x_{i,u} &= 1 \\
    y_{(i,j),(u,v)} &= 1 \\
    x_{j,v} &= 1
\end{align*} \]
• **Response Time**
  max end-to-end delay between sources and destination

• **Application Availability**
  probability that all components/links are up and running

• **Inter-node Traffic**
  overall network data rate

• **Network Usage**
  in-flight bytes
  \[ \sum_{\text{links} \in \mathcal{L}} \text{rate}(l)\text{Lat}(l) \]
ODP: Some QoS Metrics

Tunable knobs to set the optimal placement goals

\[
\max_{x, y, r} F(x, y, r)
\]

subject to:

\[
r \geq \sum_i \sum_u \frac{R_i}{S_u} x_{i,u} + \sum_{(i,j)} \sum_{(u,v)} d_{(u,v)} y_{(i,j),(u,v)} \quad \forall p \in \pi_G
\]

\[
a(x, y) = \sum_i \sum_u a_{u} x_{i,u} + \sum_{(i,j)} \sum_{(u,v)} a_{(u,v)} y_{(i,j),(u,v)}
\]

\[
B_{(u,v)} \geq \sum_{(i,j)} \lambda_{(i,j)} y_{(i,j),(u,v)} \quad \forall u \in V_{res}, v \in V_{res}
\]

\[
\sum_i C_i x_{i,u} \leq C_u \quad \forall u \in V_{res}
\]

\[
\sum_u x_{i,u} = 1 \quad \forall i \in V_{dsp}
\]

\[
x_{i,u} = \sum_v y_{(i,j),(u,v)} \quad \forall (i, j) \in E_{dsp}, u \in V_{res}
\]

\[
x_{j,v} = \sum_u y_{(i,j),(u,v)} \quad \forall (i, j) \in E_{dsp}, v \in V_{res}
\]

\[
x_{i,u} \in \{0, 1\} \quad \forall i \in V_{dsp}, u \in V_{res}
\]

\[
y_{(i,j),(u,v)} \in \{0, 1\} \quad \forall (i, j) \in E_{dsp}, (u, v) \in E_{res}
\]
The placement problem is **NP-hard**: does not scale well!

We need some **heuristics** to compute the placement in a feasible amount of time.
Heuristic: Reduce Inter-node Traffic

Aniello et al.: co-locate pairs of communicating tasks on the same computing node as to minimize inter-node communication and balance process CPU demand

Greedy heuristic – Key idea:

• Rank task pairs according to the exchanged traffic

• For each pair:
  – If node pairs a have not been yet assigned assign them to the same process
  – If either is assigned, consider least loaded node and those where they have been assigned. Work out the configuration which minimize the inter-process traffic

Xu et al. use a similar idea but assign tasks in isolation

L. Aniello, R. Baldoni and L. Querzoni, Adaptive online scheduling in storm, DEBS '13
Using ODP, we can evaluate how good the scheduling algorithm by Xu et al. works.
Heuristic: Reduce Network Usage (1)

The network usage metric combines link latencies and exchanged data rates among DSP operators: \( \sum_{\text{links } l} \text{rate}(l)\text{Lat}(l) \)

Pietzuch et al. rely on the **spring relaxation** idea:

- Application regarded as a system of springs
- Spring **minimum energy configuration** corresponds to DSP application which minimizes network usage

**Idea**: Approximate optimal placement with a **cost space**

1. Build metric cost space to encode current network latencies
2. Find query with minimal network usage in cost space
3. Map query back to physical Internet nodes and instantiate

Operator placement using a metric cost space

- Decentralized framework for minimizing network impact
- Adaptive to change in network conditions
1. Represents the application as an equivalent system of springs
Network of springs tries to minimize potential energy $E$

$E = \sum_{l \in L} DR(l) Lat(l)^2$

Streams as springs, that restore a force $F = \frac{1}{2} \cdot k \cdot s$:
- $k$ (spring constant) is the exchanged data rate on link
- $s$ (spring extension) is the latency on link

P. Pietzuch et al., Network-aware operator placement for stream-processing systems. ICDE ’06
Heuristic: Pietzuch et al.

3. Maps its decision back to physical nodes
ODP as benchmark

Distributed placement heuristic that minimizes network usage

Pietzuch et al.: \[
\min EE(y) = \min \left( \sum_{(i,j)} \sum_{(u,v):u \neq v} \lambda_{(i,j)} d_{(u,v)}^2 y_{(i,j),(u,v)} \right)
\]

Rizou et al. optimize **network usage** \((U)\)

- To solve the application placement problem, every operator optimizes its local position
- Eventually, every operator will be placed in a local optimal position
- An all-local optimal solution will also lead to a global optimal placement

\[
U = \sum_{l \in L} DR(l) \text{Lat}(l)
\]

\(U\) is a convex function,
minimum exists in a continuous space

ODP as benchmark

Distributed placement heuristic that minimizes network usage

Rizou et al.:\[ \min \ N(y) = \min \sum_{(i,j)} \sum_{(u,v):u \neq v} \lambda_{(i,j)}d(u,v)\gamma_{(i,j),(u,v)} \]

Operator placement and replication

Typical stream processing workloads are characterized by:

- high volume
- high production rate

Exploit **data parallelism**: concurrent execution of multiple operator replicas on different data portions

![Diagram of data processing with nodes labeled sources of words, count words, intermediate rank, total rank, and colors indicating medium load, high load, and overload.]
Operator placement and replication
ODRP: Opt. DSP Replication and Placement

We propose ODRP

- Centralized policy for optimal replication and placement of DSP applications
- Formulated as Integer Linear Programming (ILP) problem

Our goals:

- To jointly determine the optimal number of replica and their placement
- To consider multiple QoS attributes of applications and resources
- To provide a unified general formulation
- To provide a benchmark for heuristics
ODRP: Opt. DSP Replication and Placement

Optimal DSP Replication and Placement
• extends ODP
• jointly optimizes operator replication and replica placement

\[ x_{i,u} = 1 \]
\[ y_{(i,j),(u,v)} = 1 \]
\[ x_{j,v} = 1 \]

• U and V are multiset of computing nodes
ODRP: Opt. DSP Replication and Placement

```
Redis  ->  parser  ->  filterByCoordinates  ->  countByWindow  ->  partialRank  ->  globalRank
```

Graphical representation of the process:
- **computeRouteID**
- **source**
- **operator**
- **sink**

Bar chart showing response time versus source data rate (tuples/s):
- S-ODP_R
- S-ODRP_R
Self-adaptive Deployment

- Many factors may change at runtime:
  - load variations
  - QoS attributes of resources (e.g., cost, availability)
  - QoS attributes of network resources (e.g., latency)
  - node mobility
  - …

- How to adapt the placement and replication when changes occur? Self-adaptive deployment
Self-adaptive Deployment

- **MAPE** (Monitor, Analyze, Plan and Execute)

![Diagram of MAPE cycle]

- **Plan** phase: how to **reconfigure** the application deployment.

Possible approaches

1. Mathematical programming
2. Threshold-based approaches
3. Time series analysis
4. Reinforcement learning
5. Control theory
6. Queuing Theory
7. ...
Reconfiguration Challenges

• Reconfiguring the deployment has a non negligible cost!
• Can affect negatively application performance in the short term
  – Application **freezing** times caused by operator migration and scaling, especially for **stateful** operators

⇒ Perform reconfiguration **only when needed**
⇒ Take into account the **overhead** for migrating and scaling the operators
Threshold-based Approach

• **Upper Threshold:**
  “If CPU utilization of host is larger than $x$ for $y$ seconds, operator is overloaded.“
  \[\Rightarrow\text{Increase number of tasks by one}\]

• **Lower Threshold:**
  “If CPU utilization of host is small than $z$ for $w$ seconds, operator is underloaded.“
  \[\Rightarrow\text{Decrease number of tasks by one}\]

• **After a increase/decrease, grace period**
  – Avoid costly oscillations
We develop mechanisms for elastic stateful migration in Apache Storm.
EDRP: Elastic DSP Replication and Placement

• Unified framework for the QoS-aware initial deployment and runtime elasticity management of DSP applications

• We model *reconfiguration costs*
  – Related to migrating or scaling in/out the operators

• Centralized policy formulated as Integer Linear Programming (ILP) problem

V. Cardellini, F. Lo Presti, M. Nardelli, G. Russo Russo, Optimal operator deployment and replication for elastic distributed data stream processing, under review, 2017
EDRP Performance

• DEBS Grand Challenge 2015: popular taxis route in NYC

Minimization of response time and cost

(no reconfiguration costs)

Minimization of response time and cost
(with reconfiguration costs)
Limitations so far…

- **Centralized Optimization Algorithm does not scale**
  - EDRP is NP Hard Problem with exponential number of variables!
  → Heuristics!

- **Centralized MAPE Architecture does not scale**
  - Distributed Components but logic still centralized
  - Fully distributed solutions have limitations

V. Cardellini, F. Lo Presti, M. Nardelli, G. Russo Russo, Optimal operator deployment and replication for elastic distributed data stream processing, under review, 2017
No sooner said than done!

- Decentralize the MAPE pattern: many solutions, each with pro and cons

Hierarchical MAPEs in Storm

- New components in Apache Storm to realize the 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: monitors utilization and issues request to:
  1. Migrates an operator replica when a node is overloaded
  2. Scales out on operator overload
  3. Scales in on operator underload
     – Threshold based policies

• Application Manager
  – Generates reconfiguration tokens based on application performance
  – Grants as many reconfigurations as available tokens
     • Prioratizing scale-out > migration > scale-in
Preliminary Results

1. All reconfigurations allowed
2. Only if latency > 50% $R_{\text{max}}$
3. Only if latency > 75% $R_{\text{max}}$