Scalable Edge Computing for Low Latency Data Dissemination in Topic-based Publish/Subscribe

Shweta Khare, Hongyang Sun, Aniruddha Gokhale, Xenofon Koutsoukos and Hamzah Abdel-Aziz

Vanderbilt University
Nashville, USA

Kaiwen Zhang
École de technologie supérieure
Montreal, Canada

Julien Gascon-Samson
University of British Columbia
Vancouver, Canada
Latency Critical IoT Applications

❖ Internet of Things (IoT):
  ➢ Interconnection via the Internet of computing devices embedded in everyday objects.
  ➢ Smart City Applications

❖ Real-time New York Taxi Service
  ➢ Location and time sensitive information
  ➢ Sub-second delivery time requirement
Requirements

- Scalable Data Dissemination
- Low Latency Processing
Publish Subscribe for Scalable Data Dissemination
Edge Computing for Low Latency Processing

- Computation near the source of data on low-cost edge devices or micro data-centers.
  - Resource-Limited

Source: BI Intelligence, 2016
Publish-Process-Subscribe: Publish/Subscribe + Processing at the Edge
Latency Assurance in Existing Publish Subscribe Systems

- Widely used, open-source systems don’t provide any latency assurance.
How can we provide latency QoS assurance for publish-process-subscribe systems?
Data Driven Approach towards Latency QoS Assurance

Learn a Latency Model for Broker Load

Latency QoS is specified as a topic’s 90th percentile latency
Contributions

❖ Sensitivity Analysis
To study the impact of pub/sub features on a topic’s latency.

❖ $k$-colocation Latency Model
For predicting the latency of a topic co-located with $k$ other topics.

❖ $k$-colocation Topic Placement Problem
Place upto $k$ topics at a broker in a latency aware manner while also minimizing the number of brokers used.

$K$ is the degree of co-location of topics at a broker
Sensitivity Analysis of Pub/Sub Features

- Number of Subscribers
- Number of Publishers
- Publishing Rate
- Per-sample processing interval
- Impact of co-location/Background Load

To identify the dominant Pub/Sub features for the latency model
k-Colocation Latency Prediction Model

- **Selected Pub/Sub Features from Sensitivity Analysis:**
  - Publishing rate
  - Per-sample processing interval

- **k-colocation Latency Model input features:**
  - Features characterizing foreground topic
  - Features characterizing background load
For $k>1$, Neural Network Regression was used to capture the non-linear impact of background load on a topic’s latency.

For all $k$, the accuracy of the learned models was $\sim 97\%$. 
Prediction Model Inaccuracies

6-colocation Model

False Positives result in inefficient use of system resources.

False Negatives result in QoS violations.
**k-**\-Colocation Model Limitations

- Inaccuracy in the latency model results in QoS violations

Our approach does not provide hard guarantees on QoS assurance.
Given $k$, find a placement of topics on brokers such that latency QoS of all topics is satisfied while making minimal use of system resources.
**k-Colocation Topic Placement Heuristics**

*k-Colocation topic placement problem is NP-hard for k>=3*

- **First-Fit Decreasing (FFD)**
  Inspired by bin packing

- **Largest Feasibility Set (LFS)**
  Inspired by set-cover

- **Hybrid (LFS+FFD)**
  Combination of LFS and FFD
Comparison of Placement Heuristics

**Average 90th Percentile Latency**

- LFS<sub>k</sub> yields a lower average 90th percentile latency for all values of n.

**Percentage of topics with missed QoS**

- LFS<sub>k</sub> yields a lower percentage of topics with missed QoS in most cases.

We are able to meet the QoS for at least 87% of topics in the system.
Lessons Learned

❖ Performance of $k$-Topic Co-location heuristics relies on the accuracy of the latency prediction model
  ➢ Investigate more advanced machine learning algorithms

❖ Incorporate temporal dynamics and network link state

Thank you.
EXTRA SLIDES
Latency Assurance in Existing Publish Subscribe Systems

❖ Peer-to-Peer Pub/Sub Systems
   ➢ Re-routing paths for data-delivery
   ➢ Network level resource reservation

❖ Don’t consider processing load at the broker

System Description

❖ **System Architecture:**
  ➢ ZMQ Java sockets library
  ➢ Apache Zookeeper service for distributed coordination
  ➢ Stress-ng for broker load emulation

❖ **Dataset:**
  ➢ New York TLC dataset on taxi pickups and drop-offs.
  ➢ RIoTBench Stream processing benchmark on TLC dataset [1]: 10ms-40ms processing interval

Sensitivity Analysis - Subscription Size

Isolated Topic

Latency is below sub-second deadline for upto 300 subscribers

Co-located Topic

Latency is below sub-second deadline even with broker CPU saturation
Sensitivity Analysis- Publishing Rate

Latency increases linearly up to a threshold rate of publication, after which the increase is exponential.

Threshold rate of publication decreases with increasing background load.
Isolated Topic Latency Prediction Model

- 4 degree Polynomial Regression
- Training accuracy: 97.5%. Test Accuracy: 97%
$k$-Colocation Latency Prediction Model

Learned model does not suffer from over-fitting or under-fitting
First Fit Decreasing (FFD\(_k\))

**Algorithm 1: FirstFitDecreasing (FFD\(_k\))**

**Input:** Collection \( T = \{t_1, t_2, \ldots, t_n\} \) of \( n \) topics, latency \( \ell_i \) for each topic \( t_i \in T \) when assigned to a broker in isolation, degree of co-location \( k \), and feasibility function \( F \)

**Output:** A partition of topics \( \{T(b_1), T(b_2), \ldots, T(b_{|B|})\} \) for a set \( B \) of brokers with each broker \( b_j \in B \) hosting a subset \( T(b_j) \subseteq T \) of topics.

1. Sort the topics in decreasing order of latency when assigned to a broker in isolation, i.e., \( \ell_1 \geq \ell_2 \geq \cdots \geq \ell_n \).

2. Initialize \( |B| \leftarrow 0 \);

3. For topic \( t_i \) \((i = 1 \ldots n)\) do:
   a. \( \text{mapped} \leftarrow \text{false} \);
   b. For broker \( b_j \) \((j = 1 \ldots |B|)\) do:
      i. If \( |T(b_j)| = k \) then
          1. Continue;
      ii. If \( F(T(b_j) \cup \{t_i\}) = 1 \) then
          1. \( T(b_j) \leftarrow T(b_j) \cup \{t_i\} \);
          2. \( \text{mapped} \leftarrow \text{true} \);
          3. Break;
   c. If \( \text{mapped} = \text{false} \) then
      i. \( |B| \leftarrow |B| + 1 \);
      ii. Start a new broker \( b_{|B|} \) with \( T(b_{|B|}) = \{t_i\} \);

1. Sorts topics in decreasing order of latency when they are assigned to a broker in isolation.

2. Places the topic on the first broker that can feasibly host it along with already existing topics at the broker.

3. If no feasible broker is found, it starts a new broker/bin and assigns the topic to it.
LFS$_k$ is able to find a placement which uses less number of brokers than FFD$_k$ and LFS$_k$+FFD$_k$

LFS$_k$ takes a much longer time to find the placement solution in comparison to FFD$_k$ and LFS$_k$+FFD$_k$