Co-scheduling Feature Updates and Queries for Feature Stores

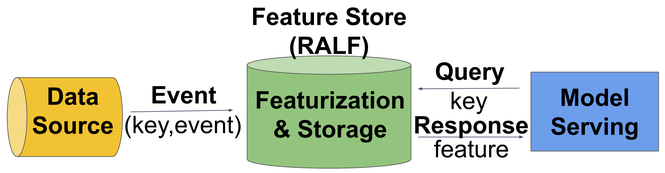
Woosuk Kwon, Debbie Liang, Jimmy Xu

riselab UC Berkeley

Goals: Design an SLO-aware featurization scheduler for online ML feature serving.

Background

Feature store: an emerging system layer that stores and updates features used for ML model training and inference.
RALF: a prototype feature store that intelligently controls feature update frequency based on the downstream feedback (e.g., model accuracy). RALF is responsible for both maintaining features and responding to feature queries in low latency.



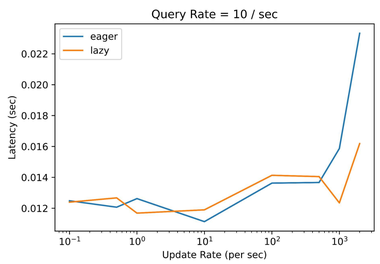
Problem: RALF does not consider query SLOs (10-100 ms) for scheduling queries and features updates. As a result, they lose opportunities to return more up-to-date features by co-scheduling query responses and feature updates.
Our solution: We designed an **SLO-aware featurization scheduler** that prioritizes updates for the features being queried over those not being queried. The scheduler improves downstream model **accuracy** by serving fresher features while meeting query SLOs.

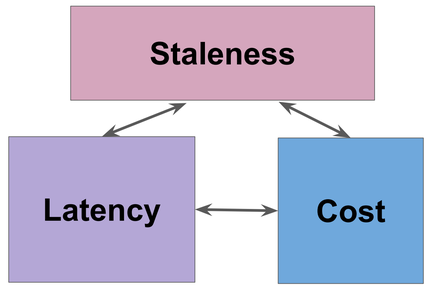
System Design 1: Exploring Trade Offs

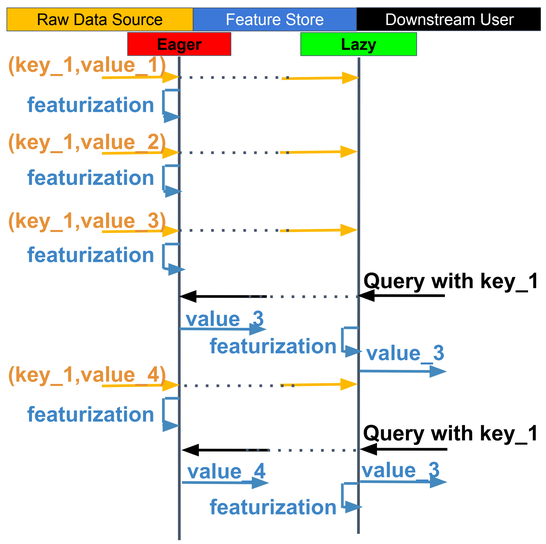
We ran multiple experiments on synthetic data to explore the relationship between eagerness and laziness in terms of

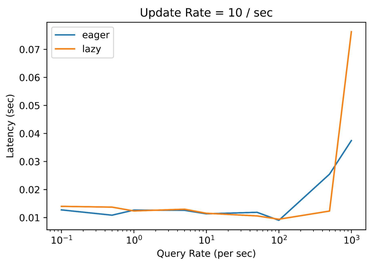
(1) **Latency** = response timestamp - query timestamp
(2) **Staleness** = response timestamp - update timestamp
(3) **Compute Cost** = number of times the featurization method was called

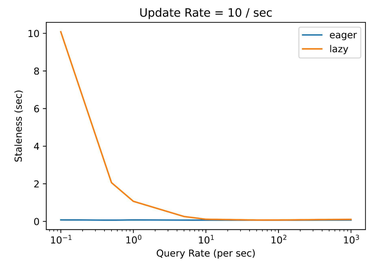
· **Laziness**: The system won't featurize the updates unless the client queries for it. Upon receiving a query, the system will either return existent value and schedule an featurization or wait for the featurization if it encounters the key for the first time.
· **Load shedding**: To avoid redundant computations, our system provides an option to group N queries and only does updates once for the same key.

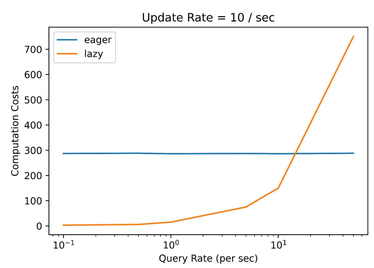






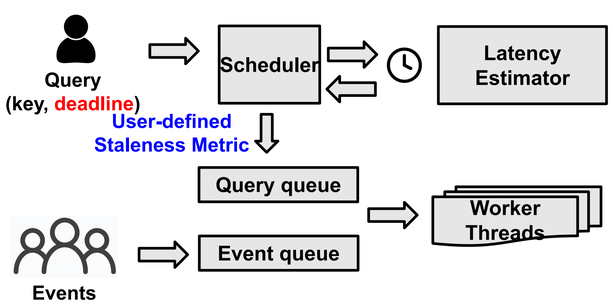




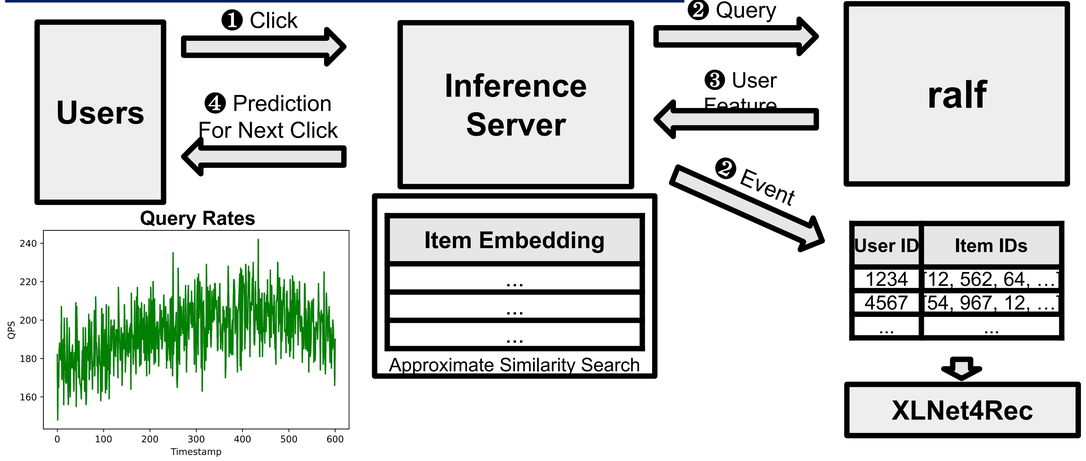


System Design 2: SLO-aware Featurization Scheduler

· **Featurization Scheduler**: Based on the explored trade offs of eager and lazy update policies, we implemented a featurization scheduler that considers query deadlines and feature staleness.
· **Latency Estimator**: The system keeps track of the states of the worker threads and profiles featurization latencies periodically.
· **User-defined query prioritization policy**: The scheduler allows users to **define their staleness metrics** to prioritize the queries to staler features over the queries to fresh features. This way, users can leverage their domain knowledge to enhance the scheduler.

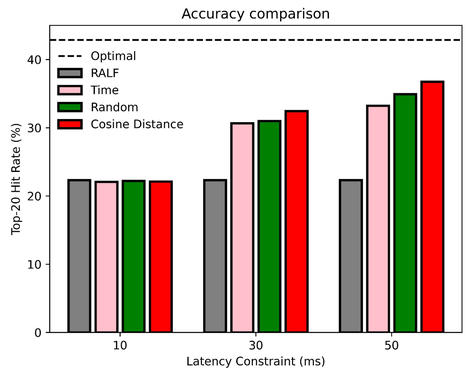


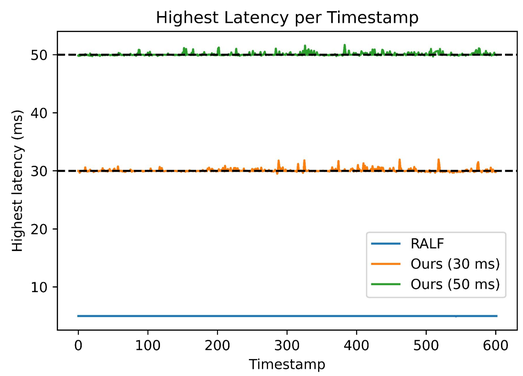
Workload: Online Recommendation Serving System



Next Click Prediction
· For every user click, the system should select 20 items the user is likely to click next.
· User feature encodes user click logs and is used for making predictions.
Challenge
· Recommendation quality is sensitive to feature **staleness**. The feature store should maintain fresh features to rapidly capture the change of user interests.
· The system should process numerous concurrent queries within tight **latencies** (< 100 ms).
Benchmark Information
· Base workload: REES46 E-commerce dataset (Oct 2019)
· Peak 10-minute browsing data is extracted
· 400K items, 30K users, 110K queries

Evaluation





*__Hit Rate__: The probability that the ground truth item is in the top 20 recommended by the model.
*__Latency Constraint__: Deadlines specified by users. (10 / 30 / 50 ms)

Key Results
1. For all 3 staleness metrics, downstream **accuracy** is considerably improved compared to RALF.
2. Using the **staleness** metric based on the domain knowledge leads to the best **accuracy**.
3. For 30 ms **latency** constraint, the scheduler achieved less than **0.3% SLO** violation rates.

Future Works

· Consider fairness in the scheduling algorithm
· Enhance predictability of featurization time for lower SLO violation rates
· Evaluate on more diverse and intensive workloads