A Self-Adaptive Fitness Club Web Application

Team 1 — Caules Ge, Ethan Li — ECE 750: Engineering Self-Adaptive Software Systems, University of Waterloo

Problem & Motivation

The Canada Fitness Club system experiences significant traffic spikes when users attempt to book courses or register during promotions. High traffic causes slow responses and increased errors, while low traffic results in wasted resources from extra replicas.

Different user devices and network conditions also create inconsistent loading times. Without self-adaptation, the system cannot stay efficient or provide a stable user experience.

Objectives

- Build a self-adaptive web application that can handle changing workloads and network performance.
- 3 levels of self-adaptation for uncertainties: Server, Application, and Client.
- Automatically scale replicas and adjust application features under heavy load, and simplify the UI during poor network conditions.
- Improve response time, reduce errors, and avoid resource waste during low-traffic periods.

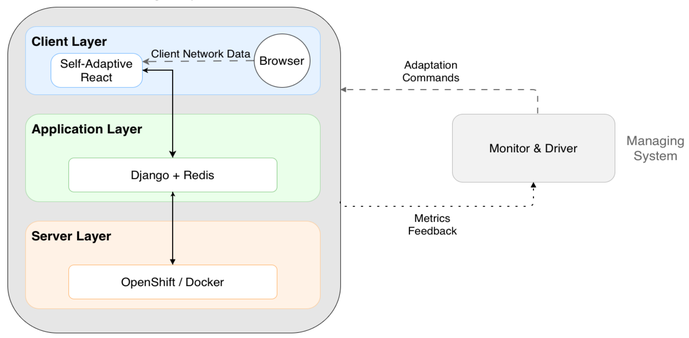
Tools & Stack

- ➢ Docker & OpenShift to deploy the application.
- ➢ Sysdig to collect core metrics from OpenShift.
- ➢ JMeter to simulate peak-traffic scenarios.
- ➢ Redis to store cached data and feature flags for adaptive behaviors.
- ➢ React to implement client-side monitoring of FPS, network quality, and rendering performance.

Server-Level Adaptation

- ❑ **Monitor:** Collect CPU, memory, and network metrics using Sysdig.
- ❑ **Analyze:** Detect high load or low utilization based on real-time metrics and the utility function.
- ❑ **Plan:** Decide whether to scale up, scale down, or enter degraded mode.
- ❑ **Execute:** Adjust replica count on OpenShift and trigger degraded mode if needed.

MAPE-K Based System Architecture



Application-Level Adaptation

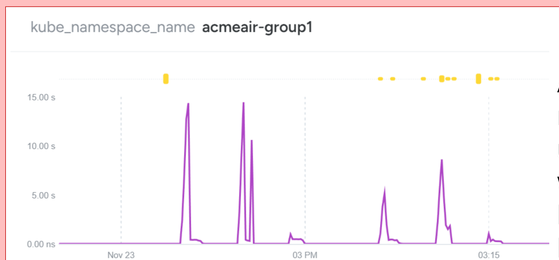
- ❑ Computes P95 latency in monitor.py and uses it to drive scaling/degraded-mode decisions.
- ❑ Uses Redis to store degraded-mode flags and cached degraded-mode responses (e.g., studio list).
- ❑ When the application reaches the scaling limit, it enters degraded mode, which reduces pagination size, disables heavy-calculated user-specific functions, and caches the page.

Server-Level Utility Function

$$
\begin{aligned}
f = \max\Bigg(0, 0.20\left(1 - \min\left(\frac{\text{ResponseTime}}{1000}, 1\right)\right) \\
+ 0.15\left(\frac{\text{TPS}}{900}\right) \\
+ 0.10(1 - \text{CpuUsed}) \\
+ 0.35(1 - \text{ErrorRate}) \\
+ 0.20\left(1 - \min\left(\frac{P_{95}}{2000}, 1\right)\right)\Bigg)
\end{aligned}
$$

The weights (e.g., 0.5 for ErrorRate) represent the relative importance of response time, throughput, CPU usage, error rate, and P95 latency.

Performance: Baseline vs Adaptive



Average response time under three workload levels, from high to low.

Client-Level Adaptation

- ❑ React monitors the average FPS, network quality, and kbps download speed of the client.
- ❑ When the average FPS drops or the network is weak, the UI switches to a lightweight mode with reduced pagination size, hides the map, and uses lower-resolution images.
- ❑ When conditions improve, the UI returns to the normal mode automatically.