courses-data: An Experiment in Client-Side Search Through a Medium-Sized Corpus

courses-data team, **sdf**

Abstract

This is left to-do. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum.

1. Introduction

Scottylabs is a student-run software development organization at Carnegie Mellon. One of our projects is CMUCourses, an online course catalog that tries to provide a good course search experience as one of its keystone features. This project, `courses-data`, was started the fall 2025 semester to look into improving search. In particular, as defined in the project description slidedeck:

- Figure out the most performant way to process and render data
- Optimize search and sorting

The scope of this article and the current work is to make search faster while adding some goodies like full-text and fuzzy search.

2. Motivating an alternative implementation

We first describe some features that we'd like to add to the search. Then, we describe at a high level how the search is implemented before our experiments. This will imply an alternative implementation that we will investigate in this article.

2.1 Additional Features

As a quality-of-life feature, our team wanted to add full-text search. Previously, the search only queried the course names and numbers, and adding the ability to search the course description could be helpful.

Also, we want fuzzy search because the current implementation isn't typo-tolerant.

2.2 Review of incumbent implementation

The incumbent implementation architecture is common among web applications, involving repeatedly sending requests to the server and getting responses. Naming these processes pings and pongs, we have:

1. (ping!) a user visits cmucourses.com
2. (pong!) the server sends that user some data that defines how the frontend should look
3. (ping!) that user triggers a search using the frontend
4. (pong!) the server runs the search query and sends back the result
5. the user's frontend renders that result
6. repeated from step 3, as desired

It should also be noted that the frontend (queried in steps 1 and 2) is cached onto the user's device by standard browser technology, so a ping-pong when visiting the site actually happens relatively infrequently. However, triggering a search (subsequent steps) must always trigger a ping-pong. This is a common quality of web applications: on top of the time it takes for the server to actually compute the result of a query, user actions also have to wait for the query to travel across the internet to the server before that computation can run in the first place, before finally getting a response that also needs to travel all the way back. Contrast this with a fully client-side application like a music player or a text editor. While they might still need to wait for IO, reads and writes from the physical disk connected to the computer are much faster than responses from and queries to a disk located somewhere else.

2.3 Key idea

This naturally leads to the thought: well, why don't we send the data to the client and have them do that search locally? This will make the initial step take longer, because we will have to send whatever data is needed to run searches. However, we entirely remove the need to make network requests when we search!

Is this feasible for our dataset? Many web applications are not able to implement this idea, because the data is simply too bulky to be stored by an end user. It would be a terrible idea for Google to send over the

entirety of the internet to any user who wants to submit a query[1].

Well, we've been working with a sample dataset that would contain roughly the same distribution of characters as whatever the modern true set of courses is. We want to run full-text-search on course name, number, and description, so how does our data look in this context?

Table 1. benchmarking search engine compression strategies

		duration of action				
compression	file size (MB)	receive (s)	decompress (s)	deserialize (s)	total (s) (cache hit)	total (s) (cache miss)
none	79.62	20.163	0	0.752	0.752	20.915
zlib	28.27	7.087	1.05	0.697	1.748	8.835
brotli	18.74	4.36	1.83	0.721	2.551	6.911
brotli*	18.74	4.36	0.349	0.113	0.462	4.822

[1] As an aside, there are at least two other valid reasons for forgoing this method. The remote server might be so fast compared to the user's computer that the net speed is faster if the server hosts it. Or, it's just more convenient to have a cloud-synced service where you don't have to think about how your data is stored.