zoominfo

A Modular Approach to Scraping Complex Web Spaces

Andrew Harris
Senior Manager, Software Engineering

October 26, 2023

I'm **Andrew Harris** and I'm a Senior Manager of Software Engineering at **Zoominfo**.

I work with a lot of *really talented engineers* on complex problems related to the **social business space**. My team spends a lot of time working with social media data and the open web.

At **Zoominfo** our solutions are aimed at connecting sales teams to the business they need to win, as quickly and accurately as possible.

Outcome driven engineering teams like the one at Zyte are a key part of how we do that.

zoominfo

What is a “complex web space?”

We define a **complex web space** as one where many users operate on a nearly *constant basis*. A few good examples include **social media** domains, **search engines**, large **corporate indexes.**

These entities are “**complex**” because they represent *robust resourcing* for **containing** and **making data publicly available** at scale and for observing traffic and activity patterns.

High traffic environments increase our responsibility to be good stewards of these spaces by leveraging *polite interaction* techniques.

What kind of complex web interactions?

Complex web spaces require a wide range of treatment patterns in response to desired outcomes. For my team, we sought to find a unified method for hosting the breadth of this range in one framework that could be leveraged by a diverse set of teams with varying levels of software design and development experience.

Social media interaction

As a person centric platform there is a constant need for our services to interact with social media data. We routinely observe updates to common, publicly available, social media sources to keep our data as fresh as possible.

Real time search behavior

As the world around us dynamically moves we observe trends in search response. Fluctuations in search relevance can be key indicators of a variety of helpful signals for our data and the market as a whole. Responding to these microtrends effectively is a key objective.

Observation of trend analytics

Trends across industries, companies, geographies, and professional spheres must be accurately reflected in our data assets. Our analysis teams have built a host of modular tools that leverage web extraction on a real time basis.

Trend relevant content listening

As a culmination of the previous categories - pivoting our listening engines to respond to market signal becomes a rational next step. Our infrastructure for planning and measuring those pivots is built around our understanding of extraction behavior.

zoominfo

Democratizing Modular Web Extraction

Extraction needs within complex web spaces are often just as *complex* and and far more *narrowly tailored* than the spaces themselves.

Usually, as needs for data consumption grow, so does the **complexity** of those extraction objectives. In most organizations these responsibilities require advanced infrastructure that must be *maintained* and *leveraged* by **advanced engineering** teams.

Often the knowledge needed tame these complex models *often rests* with groups *outside engineering* teams. Creating an environment in which a user - with **minimal engineering experience** - could significantly impact web extraction goals became a focus of development.

CORE PROBLEM

How can we architect a modular system for web extraction that scales for all our users?

zoominfo

Defining Shared Objectives

Our internal users shared similar speed and throughout goals for web extraction and for the majority of our teams an HTML payload was an expected response.

Generally standardized output expectations are best supported a similarly standardized range of input functionalities.

Cross functional availability of modular results enables faster development of shared solutions. By adopting standard markup style language we further lower the barrier for solutions engineering and scalable extraction.

Low-code environments supported by robust software infrastructure and an eloquent scheduling system.

zoominfo

Early Pain Points to Refine Our Understanding

Interaction with a range of complex web spaces **simultaneously** - and with the addition of a scheduling system by many users at once - significantly *increases exposure to detection.*

Respecting individual extraction limits for web spaces **while scaling** to efficient production levels demands the use of a range of technologies and engineering solutions.

No single model for extraction provides a universal access point for perfect data - **hygiene** and extraction must be balanced.

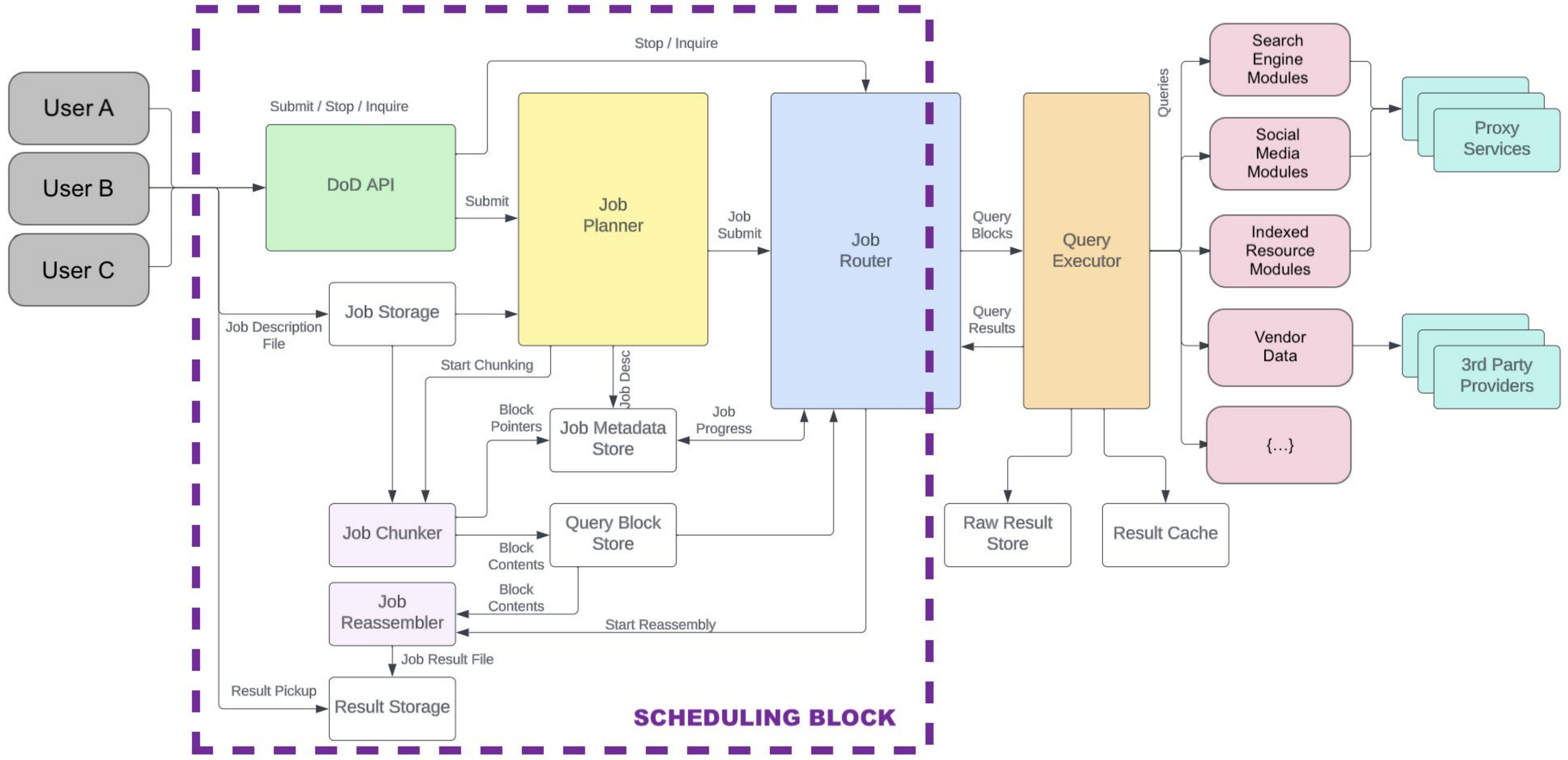
In a low code environment how a user understands **system performance** is expressed as a *function of outputs* rather than system observations.

Timing is Everything?

Hosting an extraction service for a variety of users - both human and code based - necessitates a *robust scheduling mechanism*.

Leveraging **many extraction protocols** with varying needs for call out and return sequencing does not lend well to generalization. Orchestration of load distribution for calling and processing returned data presents a resolution challenge.

Weighted queuing with *dynamic feedback* from extraction processes prevents the kinds of bottlenecks that often stall traditional concurrent crawlers. Building a solution *around* a **successful scheduling** mechanism provides a stable core for the entire infrastructure.



User A
User B
User C
Submit / Stop / Inquire
DoD API
Stop / Inquire
Submit
Job Planner
Job Submit
Job Router
Query Blocks
Query Results
Query Executor
Queries
Search Engine Modules
Social Media Modules
Indexed Resource Modules
Vendor Data
{...}
Proxy Services
3rd Party Providers
Job Storage
Job Description File
Start Chunking
Job Desc
Block Pointers
Job Metadata Store
Job Progress
Job Chunker
Query Block Store
Block Contents
Block Contents
Job Reassembler
Start Reassembly
Job Result File
Result Pickup
Result Storage
Raw Result Store
Result Cache
SCHEDULING BLOCK

zoominfo

Planning Jobs

- The **Job Planner** is a RESTful web service framework, implemented as a Java Spring Boot / WebMVC application on EKS, with pluggable modules that create execution plans from job descriptions.]
- The Job Planner processes jobs in four steps:
    - ***Planning*****,** where it selects a module based on job description and stores an execution plan for the Job Router, potentially influenced by planning hints in Job Metadata.
    - ***Chunking*****,** performed by the Job Chunker, breaks job data into blocks for later processing, with validation failure marking a job as unprocessable.
    - ***Routing*****,** following successful chunking, involves sending Job Metadata and chunks to the Job Router via stream for scheduling and execution.
    - ***Reassembly*****,** managed by the Job Reassembler, combines query blocks into a single job result file, marking the job as complete once the reassembly is finished.

Routing Jobs

- The **Job Router,** a Java Spring Boot / WebMVC application containerized on EKS, is the most complex service in the architecture, performing multiple interrelated tasks.
- Its tasks include
    - ***Scheduling*** uses a rules-based approach to allocate available capacity among jobs and participants.
    - ***Stopping*** a job involves marking it as stopped and halting the dispatch of new blocks, although currently executing blocks continue to completion.
    - ***Routing*** decides which Query Module to use for executing Query Blocks based on the current operating environment and plan from Job Planning.
- *Scheduling* employs **Weighted Fair Queueing (WFQ)** for resource allocation, ensuring fairness among teams and providing dynamic weighting for various scenarios.
- The *WFQ scheduler* iteratively monitors the work queue, selects Query Blocks for execution, evaluates the Query Plan, and sends selected blocks to the Query Executor via stream.

Executing Query Blocks

- The Query Executor, a Java Spring Boot / WebMVC application hosted on EKS, executes Query Blocks via stream, utilizing Query Modules and Result Parsing Modules.
- The Job Router maintains separate streams for each Query Module.
- When a Query Block is received via stream, the Query Executor iteratively processes each row in the block, first checking the Result Cache and using cached results if acceptable, or making RESTful web service calls to the Query Module if needed.
- It stores results in the Result Cache, streams raw data to the Raw Result Store in S3, and augments results with additional data from the cache if job hints allow.
- The Query Executor keeps track of Query Module responses and execution times, then parses the information using the appropriate Result Parsing Module.
- Once a block is complete, the results are submitted via stream to the Job Router.

Executing Query Blocks

- The Query Executor, a Java Spring Boot / WebMVC application hosted on EKS, executes Query Blocks via stream, utilizing Query Modules and Result Parsing Modules.
- The Job Router maintains separate streams for each Query Module.
- When a Query Block is received via stream, the Query Executor iteratively processes each row in the block, first checking the Result Cache and using cached results if acceptable, or making RESTful web service calls to the Query Module if needed.
- It stores results in the Result Cache, streams raw data to the Raw Result Store in S3, and augments results with additional data from the cache if job hints allow.
- The Query Executor keeps track of Query Module responses and execution times, then parses the information using the appropriate Result Parsing Module.
- Once a block is complete, the results are submitted via stream to the Job Router.

Where do we go from here?

Integrating a **machine learning layer** within the *extraction architecture* informs outcomes related to yield classification and processing of extracted data.

Similar models enable *generalization* of structured **HTML attributes** for the removal of superfluous content.

Advanced manipulation of data structures for **enhanced pattern detection** provides an advantage for detecting changes in HTML structures in complex environments.

Agnostic understanding of dynamic code structure changes in complex has become more accessible since the advent of consumer grade LLMs.