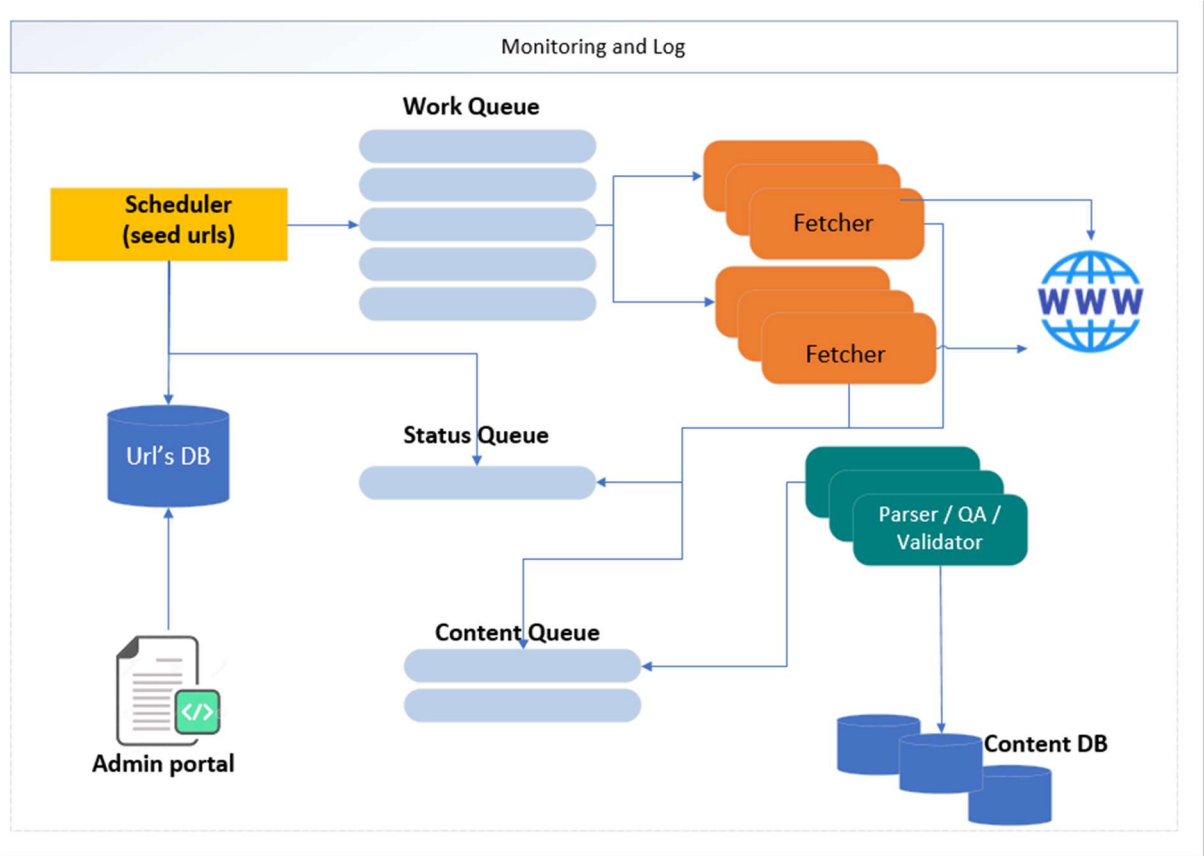
High Scale Distributed Web Scraper

Suppose you need to build a high scale distributed web scraping system. It should be able to parse about 25 million URLs at every 2 days.
Please present a cloud architecture on how you would solve this problem. List all the resources that would be necessary to implement this project and discuss about your design choices in terms of cost and efficiency.

Architecture Overview

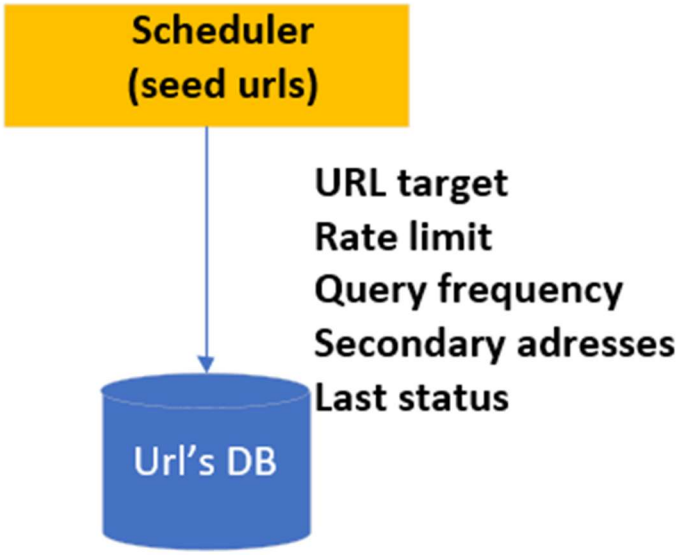


Some Requirements

- Politeness
- Scale and Distributed
- Failure prevention
- Retry police
- Async requests
- Monitoring
- Logs
- 25 million URLs at every 2 days ~ 8,680.56 requests/minute

Scheduler

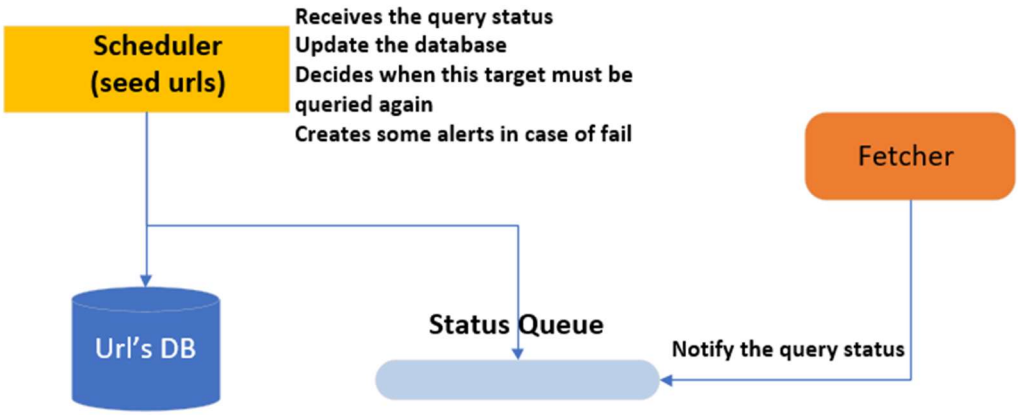
The scheduler component is responsible for to retrieve the URL's to fetch from the URL's database, analyze when it must be processed (according to the try policy) and respect the politeness and rate limiting for each target.



A web crawler can easily exhaust a website's resources within a very short period of time with negative consequences so one goal of the scheduler is to analyses the rate limit policy and the last result status for a particular target and decides if and when a particular target must continue be queried. This is one area where is necessary to do a good job right from the start.

When it's time to query some target, the scheduler sends a message to the Work Queue containing the target information, alternative target address, the retry policy and other necessary informations. This message will be processed later by the fetchers.

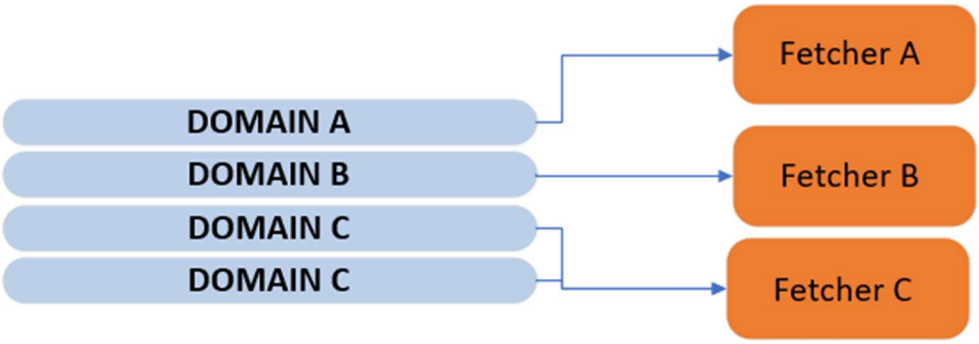
Another scheduler responsibility is to receive messages from the Status Queue and update the URL's database. This process is important because some alerts can be generated if some target is down or if the rate limit is reached.



Work Queue

In a multi-threaded environment, it should be careful to prevent race conditions when it comes to tracking requests and rate limit parameters. This problem can arise when a typical website is usually being crawled by multiple bots at the same time and to make sure that there is always an appropriate delay between consecutive queries to a website.

To implement this kind of behavior the Work Queue is designed in a way that each domain has its own dedicated queue and the message broker must onlye delivery new messages from a particular queue when there is no messages in this queue being processed (to implement this some acknowledgements mechanism is necessary).



The queue can be implemented using one of the following technologies:

- Kafka
- Rabbitmq
- Amazon Simple Queue Service (SQS)
- Azure Queue

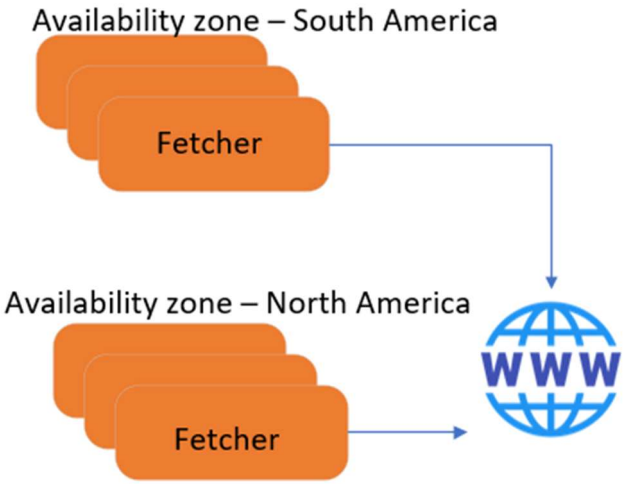
Fetcher

The fetcher component is responsible for to receiving the target URL's from the Work Queue, query the data source over the internet or using a library that encapsulates data access. if the result is completed successfully the fetcher will post the result in the Content Queue and notifies the result status through the Status Queue.

If something goes wrong the fetcher is able to understand what to do in that cases: respect the retry policy and try again with some delay, query an alternative data source or generate some alert.

Most time of the fetcher's is spent waiting for the HTTP response, so to improve the throughput it's recommended to use a technique called asynchronous programming that doesn't block the thread while the HTTP response is not received.

Some websites have some kind of IP based blocking involved or IP rate limit, so it's necessary to implement some kind of Proxies, Rotating IP solutions or distribute the fetcher's in different availability zones.



Parser

The parser component contains the logic to extract the information returned by the target, validate it and write the information to the Content Database.

In the head request we have the last modified information so comparing with the last modified information saved in the database the fetcher decides if the information must be updated or not.

To ensure that the data scraped data is accurate and reliable, it's necessary to run some QA tests on it. It's important to have a set of tests for the integrity. Some of it can be automated by using regular expressions, threshold limits or check if the data follows a predefined pattern and in the worst case generate some alerts to be manually validated.

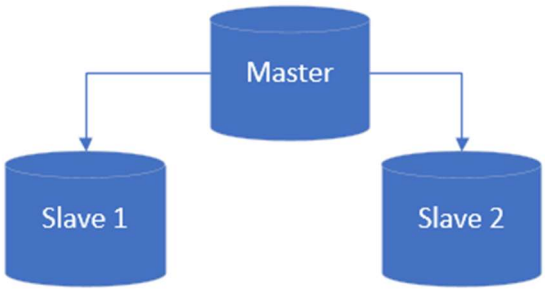
Websites will change their structure with some frequency, so usually the Parser need some adjustments frequently. It's necessary to monitor the parser to detect possible changes and fix it as fast as possible.

Content Database

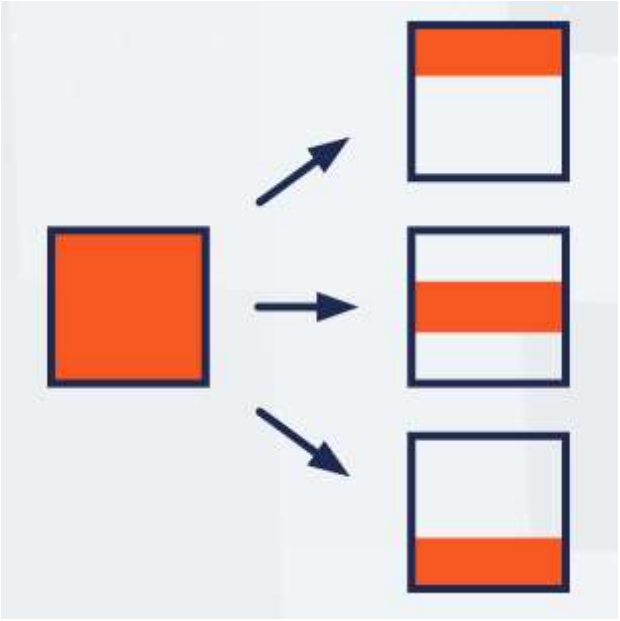
The Content Database stores the information extracted by the parser and to guarantee scalability and reliability of the system, it's inevitable that the data will be distributed in more than one place.

Every time that some data is written in the database, it's necessary to have a copy of that data in the other replicas. If a client that sends a write request to some node in the cluster, the cluster must follow some strategy to replicate this data in the other nodes.

The most common replication strategies are: replication with a single leader, replication with multiple leaders and replication without a leader.



Another technique that can be used to increase database scalability in the point of view how to query the data is data partitioning (sharding) that can be applied to very large data sets with a very high volume of queries.



The type of the database depends on what kind of data will be saved. If it's required to have some flexibility on the schema a NoSQL database can be used but if integrity is required a relational database must fit well in this case. There are many types of NoSQL databases(key-value pair, column-oriented, document-oriented) and the best choice depends on the data structure to be stored.

Admin portal

The admin portal is a web application that interact with users allowing to create new target URLs, configure when to query it, alternative addresses, the retry and alert policy and also has a dashboard containing the overall system status.

When a Fetcher or a Parser finishes being repaired the admin portal has an interface to allows to make a query to the target in a manual way.