**Web Crawler Design Doc**

*Authors:* [*S*](mailto:username@chromium.org)*huvra Kanti Nath  
May-18, 2024*

### **Summary**

I am proposing a redesign of the web crawling infrastructure to address scalability, maintenance, and monitoring concerns. The current system involves deploying individual crawlers to separate Google Compute Engine (GCE) instances for each site, resulting in high costs and operational complexity. The redesign aims to centralize code maintenance, improve scalability, and enhance monitoring capabilities.

### **Platforms**

Mac, Windows, Linux, Chrome OS, Fuchsia, Android, Android WebView, WebLayer, iOS.

### **Team**

Email: [shuvracse03@gmail.com](mailto:shuvracse03@gmail.com), Mobile:01307726542

### **Bug**

Pending.

### **Code affected**

Crawler infrastructure, monitoring systems.

# **Design**

The current system, while functional, poses challenges in scalability, maintenance, and monitoring. With over 150 individual crawlers deployed on separate GCE instances, code maintenance becomes burdensome, and monitoring the status of each instance is challenging and costly. Also, current code has new crawler for each site.

**Proposed Solutions:**

* **Generic Crawler**

I have designed a single crawler(GenericCrawler) where I specify the allowed domains and start URLs for all the sites we want to crawl. start\_urls can be read from a database or a file also, and we can add new sites there easily without declaring a new scraper. If new parsing logic is needed, we can easily add that in the single file. In this way, we shall be running only one single spider instead of 150 different crawlers

Example:

==================

import scrapy

from scrapy.spiders import CrawlSpider, Rule

from scrapy.linkextractors import LinkExtractor

class GenericCrawler(CrawlSpider):

name = 'generic\_crawler'

allowed\_domains = ['site1.com', 'site2.com', 'site3.com']

start\_urls = ['https://site1.com/', 'https://site2.com/', 'https://site3.com/']

rules = [

Rule(LinkExtractor(), callback='parse\_item', follow=True)

]

def parse\_item(self, response):

site\_url = response.url.split('/')[2] # Extracting the domain from URL

if site\_url == 'site1.com':

self.parse\_site1(response)

elif site\_url == 'site2.com':

self.parse\_site2(response)

elif site\_url == 'site3.com':

self.parse\_site3(response)

def parse\_site1(self, response):

# Parsing logic for site1.com

title = response.css('h1::text').get()

page\_url = response.url

self.save\_data(title, page\_url, 'site1.com')

def parse\_site2(self, response):

# Parsing logic for site2.com

title = response.css('h2::text').get()

page\_url = response.url

self.save\_data(title, page\_url, 'site2.com')

def parse\_site3(self, response):

# Parsing logic for site3.com

title = response.css('h3::text').get()

page\_url = response.url

self.save\_data(title, page\_url, 'site3.com')

def save\_data(self, title, url, site\_url):

# Placeholder for saving data to a database, file, or any other destination

print(f"Site URL: {site\_url}\nTitle: {title}\nURL: {url}\n")

* **Task Schedular integration**

Implement a task scheduler (e.g., Google Cloud Scheduler) to automate the monthly crawling of existing sites and the addition of new sites.

* **Centralized Monitoring**

Integrate monitoring tools (e.g., Stackdriver Monitoring) to provide real-time visibility into the health and status of the crawler from a centralized dashboard. I can even write a small dashboard using Vue JS + Flask to show the scraping info in real time by reading the scraping data.

# Metrics

## Success metrics

* Number of Successfully Crawled Sites per Month: Measure the number of sites crawled successfully each month to ensure coverage and identify any issues.
* Reduction in GCE Instance Usage and Costs: Track the decrease in the number of GCE instances used and the associated cost savings.
* Decrease in Maintenance Efforts and Time: Measure the time spent on maintenance tasks before and after the redesign to demonstrate reduced overhead.
* Improved Data Freshness and Accuracy: Track the frequency and accuracy of data updates to ensure the crawler is providing up-to-date information.
* Increase in the Number of Sites Managed Seamlessly: Monitor the addition of new sites to ensure the process is smooth and does not require extensive rework.

## Regression metrics

* Error Rates or Failures in Crawling Tasks: Monitor the rate of errors or failures in the crawling process to detect any regressions or issues.
* Resource Utilization: Track CPU, memory, and network usage to ensure there are no unexpected increases.
* Latency of Data Updates: Measure the time taken for data to be updated in the system to ensure it does not regress.
* System Response Times: Ensure the response times for crawling tasks remain optimal and do not degrade over time.

## Experiments

* A/B Testing: Compare the performance and efficiency of the new crawler infrastructure against the old system to validate improvements.
* Performance Monitoring: Conduct load tests and monitor system performance during peak crawling times to identify any potential bottlenecks.
* User Feedback Analysis: Collect and analyze feedback from users and stakeholders to assess the impact of the new crawler on their operations.

# **Rollout plan**

Waterfall: The rollout will follow a waterfall model, with thorough testing and validation at each stage before progressing to the next.

# **Core principle considerations**

## **Scalability**

## The new design supports easy addition of new sites without the need to deploy new instances or write new scrapers from scratch.

**Efficiency**

Centralized crawling reduces the overhead of managing multiple GCE instances.

**Maintainability**

The generic crawler design simplifies code maintenance and reduces duplication.

**Monitoring**

Enhanced monitoring capabilities provide better visibility into system health and performance.

# **Testing plan**

* Unit Tests: Implement unit tests for individual parsing functions for each site to ensure correctness.
* Integration Tests: Conduct integration tests to ensure the crawler works seamlessly with the task scheduler and monitoring tools.
* Load Tests: Perform load tests to verify the crawler’s performance under different loads and identify any performance issues.
* Regression Tests: Execute regression tests to ensure the new system does not introduce any new issues and maintains existing functionality.

# **Followup work**

* Success Assessment: Regularly assess the success of the new crawler based on the defined success metrics.
* System Monitoring: Continuously monitor the system for any issues or performance bottlenecks.
* Refinement and Expansion: Continuously refine the parsing logic and add new sites as needed.
* Feedback and Adjustments: Evaluate feedback from the monitoring tools and make necessary adjustments to improve system performance.
* Cleanup: Clean up any experimental code or features that were used during the testing and rollout phase to ensure a stable and maintainable codebase.