Parallel Web Crawler

Syed Ali
CS 4438
Jan 28, 2019

ABSTRACT

We implement a parallel web crawler using Python. The web crawler conducts a breadth-first search from any origin URL and continues until it has reached a pre-set maximum number of elements, or until a timeout occurs. Timeouts are linked to performance issues – a number of threads are used to expand the boundary of the breadth first search at every iteration, and if a timeout occurs, more time or threads are needed. The number of threads and timeout seconds can be modified, as per the readme.md document, which user should look at before using this web crawler.

INTRODUCTION

Modern web crawlers work form the backbone of web search. As the web moves towards ontological structure with web ontology languages and more, web crawlers remain essential to retrieving and organizing information.

The problem studied is described at [1]. The primary goal is to construct a graph theoretic representation of the web by scraping websites for hyperlinks to other websites to create a directed graph. The secondary goal is to compute some statistics about the constructed graph, including the graph's diameter, the average distance between two nodes, the number of strongly connected components, and the destruction of incoming and outgoing links to and from each node.

Nodes represent pages on websites which are reached by URLs, so we present each node by the string corresponding to it's URL.

IMPLEMENTATION

A standard breadth first search is applied, where the boundary is kept track of, however unlike a standard breadth first search, we also store the hyperlinks in a two-way mapping. The two-way mapping maps from site's which link to and from other URLs. For example, if a site A linked to a site B, we would track an entry in a hyperlinks_to mapping which would map A to B, and a hyperlinks_from mapping, which would map B to A. This allows us to construct a richer representation of the graph, should we want to extract additional statistics or run algorithms to further analyze the graph.

Additionally, on each iteration of the breadth first search, there are n threads put together in a computation pool to retrieve and scrape each URL for other URLs, where n is the maximum of the number of URLs in the current boundary of the breadth first search, and some contact max_n

In order to compute statistics, we take advantage of [2], NetworkX. This library helps us run algorithms like Tarjan's strongly connected components algorithm, compute the diameter, etc.

CONCLUSIONS

The web crawler function well at a small scale, but struggles as the boundary expands and the timeout must grow very high to accommodate the explosion of linked sites. Growing the number of threads is not recommended, as it may bring the machine running the crawler to a halt, as it struggles to allocate enough CPU time to each thread.

This parallel web crawler could be improved by dynamically allocating threads to perform scraping based on the performance of previous threads, so that too many or too few threads are not allocated, based on the machine. This issue is fundamentally one of machine variability: the same number of threads running on one machine may bring another machine to a halt.

Accordingly, another improvement would be to scrape the same number of URL in each iteration, so that the pool does not become more likely to timeout as the boundary expands. A simple solution would be set the limit of crawling at say, 100 URLs at a time, and to feed the thread pool with the same number of URLs so that URLs which are closest to the origin are fed first, while others are fed later, so that the uniformity of the boundary in the breadth first search is maintained.

There is also some minor instability with the web crawler, however this is largely due to sites closing connections or acting oddly in some circumstances, particularly when many requests are made, suggesting that these sites are ignoring or blocking requests from the crawler. Improvements in this area would allow for high volume queries, but may not be desirable, as they would increase the load on site servers.

REFERENCES

[1]R. Solis-Oba, "Projects", Csd.uwo.ca, 2019. [Online]. Available: http://www.csd.uwo.ca/courses/CS9668b/projects.html. [Accessed: 31- Jan- 2019].

[2]"NetworkX — NetworkX", Networkx.github.io, 2019. [Online]. Available: https://networkx.github.io/. [Accessed: 31- Jan- 2019].