Web-crawler development

Thoughts

* **Some design choices :**

1. Storing all URLs related to Job URLs

I chose to keep trace of all the URLs, including their data (images), found when recursively crawling each Job URLs (the Urls defined for a Job) in case we would add a new entry point asking to display every URL links crawled from each specified URL.

It could also be useful if the entry point asking for the Job status would actually need to display the whole number of waiting URLs and completed URLs, indeed including all the URLs found during crawl processing.

For information, quick overview of the data structures:

*allJobs* (Jobs structure) is the datastore parameter when running the API containing each added job.

In particular, each Job has four structure parameters, and for simplicity, three out of them of are exactly the expected JSON formats required by the entry points.

The fourth one is of *JobProcess* structure type: this one is being updated during the whole processing of the Job:

* The parameters *urlsProcesses* manages the data for each Job URLs.

It is a Map such as:

* + Keys are the Job URLs (similar as *Urls* from *JobDef*),
  + Values are *UrlProcess* structure type essentially containing the following parameters:
    - *WaitingUrls*: all the URLs got in relation with the Job URL crawling ready to be crawled. This set is initialized to the Job URL.
    - *ProcessingUrls*: all the URLs from the WaitingUrls set that are being crawled, and for which the data (images) are being added,
    - *CrawledUrls*: all the URLs from the *ProcessingUrls* set that have been crawled.

1. Updating Job status only on request

*UpdateJobStatus* method, responsible for computing the number of in\_progress and completed Job URLs, is processed only in the following conditions:

* + on request when accessing the entry points */jobs/{job\_id}/status* or */jobs/{job\_id}/result*,
  + in the *WorkOnJob* method for the workers to check if the job is done (indeed if all the Job URLs are completed).

I did not consider interesting to do this update each time an URL is crawled to save CPU consuming.

In particular, due to the design choice explained in point 1-, necessary merges are done for each Job URL, to merge all the data of all the related URLs.

* **An interesting challenge:**

Workers goroutines: how to maximize efficiency of the workers/goroutines for each job ?

Sometimes, no URL are available (in *waitingUrls* parameter), which does not always mean that the job is done: in particular, at the beginning of the job, only the specified URLs are available, and if there are more workers than the number of specified URLs, some of them could not have any URL to crawl, so as soon as it happens they leave the job by ending their goroutine.

So the challenge was to find a way to keep them alive waiting for new URLs to work on.

Many ideas have been tried such as:

* + *Use of a buffered channel for each worker activity*:

Idea: the workers could leave the job only when the channel is full, meaning that all the workers are inactive.

Result: not working because every data transmitted on this channel is irreversible, and one worker could easily fill the buffer when alternating between cycles being active on task and cycles being inactive waiting for task to do.

* + *Use of a timer / number of waiting cycles*:

Idea: the worker could leave the job after some time passed while waiting for a task to do.

Result: not working because not deterministic, as the time depends a lot on how long the other workers take to do their crawl and give new URLs to crawl.

* + *Use of a job activity flag*:

Idea: shared among the workers, this flag could be set to true by any active worker, and set to false by any inactive worker.

Result: not working, because it is assumed that if the job is not done, as soon as one inactive worker would set the flag to false, another active worker would set it back to true.

But in reality, an inactive worker setting this flag to false would immediately see this flag as false and consider that the job is done.

* + *Use of workers’ status*:

Idea: same as the job activity flag, but instead of considering a unique flag, every worker has his own status flag. Only when all the flags are set to false, the job would be considered as done.

Result: working. That way, all the workers leave at the same time at the very end of the job, indeed when there are no more waiting URLs.

* + *Use of a processing URLs set*:

Idea: this set allows tracking the URLs that are being crawled, which is very useful to complete the waiting URLs set.

That way, even if at some moment the waiting URLs set is empty, the job can still remain considered as not done if at least one URLs is being crawled, indeed belongs to the processing URLs set.

Result: working, implemented. That way, all the workers that do not see any waiting URL to crawl, ask for the job status to see if some URLs are still being crawled, and can leave at the very end of the job, indeed when there are no more waiting URLs nor processing URLs.

* **Possible improvements:**

1. Error catching, when encoding/decoding a JSON, or when reading a URL content could be better managed if all the possible issues could be known and then anticipated.
2. Crawling algorithm (*CrawlUrl* method) could probably be reworked to avoid all the if-else nested.

The way the URLs links are identified in an HTML content could also be improved: for example, “href” tag could be in other tokens than “a” or “link”.

1. The port of the URL connection could be passed as an environment variable set when running the Docker container, rather than having it hard-coded in the application.