Web-crawler development

Thoughts

* **Some design choices :**

1. Keeping all children URLs per specified URL

*allJobs* (Jobs structure) is the datastore parameter when running the API, containing each added job (Job structure) and all their data.

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

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

* The parameters *waitingUrls* manages the URLs waiting to be crawled.

It is a Map such as:

* + Keys are the URLs specified during the Job definition (similar as *Urls* from *JobDef*),
  + Values are *UrlData* structure type essentially containing the following parameters:
    - *childrenUrls*: the key URL + all the URLs got during the key URL crawling, and then ready to be crawled on their own,
    - *data*: all the images got during the key URL crawling and its children.
* The parameters *completedUrls* manages the URLs already crawled.

It is a Map such as:

* + Keys are the URLs specified during the Job definition (similar as *Urls* from *JobDef*),
  + Values are all the children URLs of the key URL already crawled.
* I chose to keep trace of all the children URLs of each specified URL 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 also the children URLs found during crawl processing.

1. Updating Job status only on request

*UpdateJobStatus* method, responsible for computing the number of in\_progress URLs and completed URLs, is processed only on request when accessing the entry point */jobs/{job\_id}/status*.

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

* **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, solution implemented. 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.

* **Possible improvements:**

1. Both *waitingUrls* and *completedUrls* have the specified URLs as key, which is a non-useful redundancy that can be removed by redefining the *JobProcessData* structure.
2. 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.
3. Crawling algorithm (*CrawlUrl* method) could probably be reworked to avoid all the if-else nested.

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