Web search app

One of the original uses for the MapReduce framework is indexing the web. MapReduce can handle many of the tasks required for building a search engine:

Crawling Crawling is the process of exploring the web of links to discover a large number of web pages. Mappers can look up URLs and parse the documents to find a new set of URLs; reducers can then collect the documents and remove duplicates.

By iterating this process you can quickly find a large number of documents.

Ranking Search engines are useful because they can find the most relevant information for a given query. In order to do this, they need to rank pages by relevance.

One of the best-known algorithms for doing so is called PageRank (named after Larry Page). The basic idea behind PageRank is that you compute a "usefulness" estimate for each web page. You start by assuming that each page is equally useful. You then iteratively improve your usefulness estimates by distributing the usefulness of each page p to all of the pages that p links to. By iterating this process, the pages that are referenced from many other pages will become more useful, especially if they are referenced from other useful pages.

For more details on the PageRank, consult the PageRank article on Wikipedia, particularly the "Iterative implementation".

MapReduce is naturally suited for implementing PageRank; each iteration can be implemented as a map reduce job; the mappers will compute the contribution from each page to each other page; the reducers can sum these contributions to find the total usefulness of each page.

Indexing When a user wishes to perform a search, it is important to quickly find a list of relevant pages that match their query. MapReduce is a good way to form this index: The map phase can associate pages to queries; The reduce phase can then sort the pages relevant to a given word by rank.

Searching Depending on how you index your data, you may also find it useful to perform a MapReduce job to handle a user's query.

There are a lot of components to a fully featured search engine. You do not need to complete all of them for full credit for your elective app.

The Web module

We have provided a `Web` module that uses the `ocamlnet` library to perform basic web page lookup and parsing. It provides the `fetch_page` function, which looks up a given URL, and parses the resulting HTML page to extract the title, set of links, list of all words, and list of keywords (from the `meta` tags).

We encourage you to either use the web module as is, or modify it to suit your needs. If you wish to modify it, you may want to consult the Nethtml documentation.

To use the `Web` module, you will first have to install ocamlnet and cohttp:

```
opam install cohttp ocamlnet
```

You will also have to configure your cs3110 tools to compile with ocamlnet by adding `cohttp.async` and `netstring` to the `.cs3110` file:

```
compile.opam_packages=async,cohttp.async,netstring
```

You will also have to remove `str` fomr `compile.ocaml_libraries` since `netstring` include `str` itself. When working with the module in `utop`, you will also need to `#require "cohttp.async"` and `#require "netstring"`.

A note on big data

The web is large; too large to store an index in memory even with a large number of computers. Our implementation of this app is able to comfortably handle hundreds of pages.

You could increase this limit if you design your app to store intermediate data in files on a shared filesystem; then instead of passing actual data between workers, you can pass filenames. This will allow you to handle much bigger data sets without fundamentally changing your MapReduce infrastructure.