Task 1: Communication models and Middleware

Return to: General

The goal of this task is to understand and use communication models and middleware concepts.

Students should distinguish between the following communication models: **synchronous** and **asynchronous**, **pull** and **push**, **transient** and **persistent**, and **stateless** and **stateful**.

Students should also understand direct (rpc, distributed objects) and indirect (message queues, pub/sub, group communication) communication middleware

Playing out with MapReduce

1. Introduction

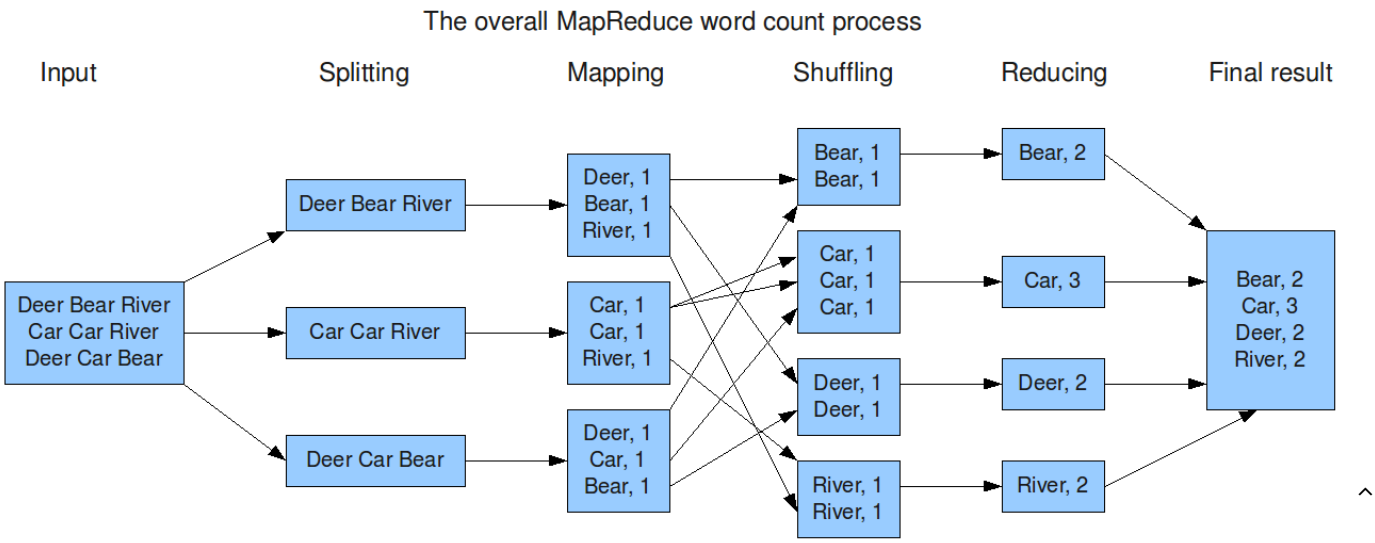
MapReduce is a programming model and implementation to enable the parallel processing of huge amounts of data. In a nutshell, it breaks a large dataset into smaller chunks to be processed separately on different worker nodes and automatically gathers the results across the multiple nodes to return a single result.

As it name suggests, it allows for distributed processing of the **map()** and **reduce()** functional operations, which carry out most of the programming logic. Indeed, it consists of three majors steps, which are the following ones:

- **"Map" step:** Each worker node applies the "map()" function to the local data, and writes the output to a temporary storage. A master node ensures that only one copy of redundant input data is processed.
- **"Shuffle" step:** Worker nodes redistribute data based on the output keys (produced by the "map()" function), such that all data belonging to one key is located on the same worker node.
- **"Reduce" step:** Worker nodes now process each group of output data, per key, in parallel.

Because each mapping operation is independent of the others, all maps can be performed in parallel. The same occurs to the reducers, given that all outputs of the map operation with the same key are handed to the same reducer.

Here you can see the general process of MapReduce for counting the frequence of each word, what is known as Wordcount. Each map phase receives its input and prepares intermediary key as pairs of (key,value), where the key is the actual word and the value is the word's current frequency, namely 1. Shuffling phase guarantees that all pairs with the same key will serve as input for only one reducer, so in reduce phase we can very easily calculate the frequency of each word.



For a general overview of MapReduce, we recommend you to read the (in)famous Google paper: MapReduce: Simplified Data Processing on Large Clusters:

https://static.googleusercontent.com/media/research.google.com/es//archive/mapreduce-osdi04.pdf

2. Goals

The objective of this practical assignment is to write two very simple programs:

1. **CountingWords**: Counts the total number of words in a text file. For example, given the following text: "I love Distributed Systems", the output of CountingWords should be **4 words**.
2. **WordCount**: Counts the number of occurrences of each word in a text file. For instance, given the following text: "foo bar bar foo", the output of WordCount should be: **bar, 2; foo, 2**.

In MapReduce systems like Hadoop, both programs are the equivalent of the standard "Hello, world!" C program you used to write when learning a new programming language.

3. Tasks to do

3.1. MapReduce prototype

First of all, you must implement a simplified prototype in **PyActor** of the MapReduce model.

More concretely, your task is to implement a simple **master/slave** architecture without considering complex issues like scalability or fault tolerance. To do so, the master component should be able to spawn **map()** and **reduce()** workers onto remote hosts by exploiting the PyActor's feature of **remote spawning** described in the **Sample 3** of the remote tutorial (http://pyactor.readthedocs.io/en/master/remote_tuto.html#sample-3-remote-spawning)

As a further simplification, you don't need also to implement the **"Shuffle"** step. Throughout this task, we will assume the existence of a single reducer. Hence, all the parallelism will be brought to the system through the parallelization of the map() operation.

Finally, you also don't need to support the automatic partitioning of input data in your code. You can manually partition the text file in N files that will then be read by the N mappers you launch. Use the following command to launch a web server in python:

```
python -m SimpleHTTPServer
```

Use the web server to distributed the text files to map actors.

3.2. MapReduce jobs

Once upon you have implemented a prototype of MapReduce in PyActor, you should implement both the **CountingWords** and **WordCount** programs.

4. Evaluation

To evaluate your solution, you should at least use one of the following datasets:

1. Sherlock Holmes (6.5M) (English): https://norvig.com/big.txt
2. El Quijote (2.2M) (Spanish): http://www.gutenberg.org/cache/epub/2000/pg2000.txt
3. The Bible (4.5M) (English): http://www.gutenberg.org/cache/epub/10/pg10.txt

Calculate the speedup of your platform compared to a sequential implementation. Make experiments with different number of mappers.

5. Deliverables

Prepare a report explaining the architecture and validation of your solution, along with your code. The report must also include the speedups achieved by your solution. The explanation should be concise but meaningful. Please, avoid "biblical" explanations that don't shed light on the internals of your solution.

Personal github. All groups must create their own github repository with their code and documentation. You must include basic code coverage and health links. This requires the development of unitary tests for your code. Example of a https://github.com/danielBCN/PyActor-example

6. Assignment scores

The corresponding scores for each task are the following:

[5 pts] General implementation of the MapReduce prototype.

[2 pts] Correct implementation of the CountingWords program, along with the corresponding speedups.

[2 pts] Correct implementation of the WordCount program, along with the corresponding speedups.

[1 pt] A good writing of the deliverable report explaining your solution: architecture, implementation and validation.

Estat de la tramesa

Estat de la tramesa	Cap intent
Estat de la qualificació	Sense qualificació
Data de venciment	dimarts, 3 abril 2018, 23:55
Temps restant	27 dies 14 hores
Darrera modificació	-
Comentaris de la tramesa	▶ Comentaris (0)

Afegeix la tramesa

Feu canvis a la vostra tramesa

Return to: General