Dr. Marc Sanchez-Artigas
marc.sanchez@urv.cat

Distributed Systems 2019/20

Task#1

Guidelines

- Please submit your solutions until **Wednesday, April 1, 23:55 PM** through Moodle. Only electronic submission is accepted. Please only use PostScript or PDF. We must be able to print out your submissions on a standard monochrome printer so that they are fully readable.
- This assignment can be solved in **teams of two students**.
- By submitting any processed exercise you hereby commit to Ethics rules promulgated by URV (see http://wwwa.urv.cat/la_urv/3_organs_govern/secretaria_general/legislacio/2_propia/comunitat/estudiants/inst_regl_reg_disc_est.pdf). This means that you should try to write the answers in your own words. However if you use external materials, you have to cite them correctly.

1. Overview

The goal of this assignment is to implement a distributed version of a simple algorithm: *matrix multiplication*, which is easier to solve.

1.1 General architecture of the system

There will be a number of **workers** ***W***, which would act on parallel on chunks of data, and an **object storage store** (e.g., IBM COS), which will store the input and output matrices.

1.2 Implementation technologies

For the implementation, you MUST use **IBM-PyWren**[1] [1]. **IBM-PyWren** is an open source project that allows the execution of Python code at scale on top of IBM Cloud Functions, i.e., IBM's Functions-as-a-Service (FaaS) platform, based on Apache OpenWhisk. PyWren delivers the user's code into the serverless platform without requiring knowledge of how functions are invoked and run. It supports a basic MapReduce API.

1.2.1 MapReduce

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.

2. Matrix multiplication

The goal of this problem is to provide a distributed implementation of the multiplication of two matrices, namely $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times l}$. For simplicity, we will implement a non-optimal version of multiplication. More concretely, each worker will fetch and multiply an $a \times n$ row-block of A and $n \times a$ column-block of B to compute an $a \times a$ block of the output matrix, namely $C \in \mathbb{R}^{m \times l}$, as illustrated in Figure 1. Note that each worker will have to fetch each row-block and column-block from the server and store the corresponding $a \times a$ block back to the server. Figure 2 contains a detailed description of the steps to be taken for matrix multiplication.

1. IBM-PyWren - PyWren on IBM Cloud, URL: https://github.com/pywren/pywren-ibm-cloud

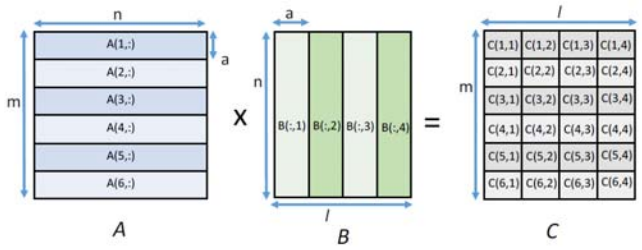

Figure 1. Graphical representation of matrix multiplication.

Input : Matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times l}$
Result: $\mathbf{C} = \mathbf{A} \times \mathbf{B}$
1 **Initialization**: Divide $\mathbf{A}$ into submatrices of size $a \times n$ (row-wise division) and $\mathbf{B}$ into submatrices of size $n \times a$ (column-wise division)
2 **for** *i=1 to m/a* **do**
3 **for** *j=1 to l/a* **do**
4 1. Worker W_{ij} receives i-th chunk of $\mathbf{A}$, say $\mathbf{A}(i,:)$, and j-th chunk of $\mathbf{B}$, say $\mathbf{B}(:,j)$
5 2. W_{ij} computes the $a \times a$ chunk of $\mathbf{C}$, that is, $\mathbf{C}(i,j) = \mathbf{A}(i,:) \times \mathbf{B}(:,j)$
6 3. W_{ij} writes $\mathbf{C}(i,j)$ back to the cloud storage
7 **end**
8 **end**

Figure 2. Detailed steps for matrix multiplication.

2.1 Tasks

Here you can find the list of tasks to implement:

1. **[7 points]** Implementation of the matrix multiplication. The code should allow changing the number of **workers *W***. Ideally, this means that your solution should adapt the subdivision of the matrices into row-wise and column-wise blocks based on the number of **workers *W***
2. **[2 points]** Experimentation with different values of m, n, l and a, to see how the matrix, row-block and column-block sizes affect the total running time. **Finally, plot the results as a function of these parameters**. As a metric for comparison, you can use the total running time for each configuration of parameters, and even the speedup. Recall that speedup is defined as the ratio of the time required to compute some function using a single processor (T_1) divided by the time required to compute it using W processors (T_W). That is: $\text{speedup} = T_1 / T_W$. For example, if it takes 10 seconds to run a program sequentially, and 2 seconds to run it in parallel on some number of processors W, then the speedup is $10/2 = 5$ times.
3. **[1 point]** Satisfactory documenting of the solution and results.

3. Submitting

Please, submit a three-page document describing your proposed solution along with the plots. Further, an in-depth explanation of the results is MANDATORY. The document should also include full references for the papers and other sources that you have consulted. **If you are working in a group, you must say who has been responsible for each portion of the work.**

Personal github. All groups must create their own `github` repository with their code and small documentation.

Finally, create a `.zip` file called `DS_A1_name1_name2.zip`, where `name1` and `name2` will be the family name of each component of the group. Include in the `.zip` file the document in `.pdf` format along with all the Python code.

References

[1] - Josep Sampé, Gil Vernik, Marc Sánchez-Artigas, Pedro García-López, “Serverless Data Analytics in the IBM Cloud”. Middleware Industry 2018: 1-8

1. IBM-PyWren - PyWren on IBM Cloud, URL: https://github.com/pywren/pywren-ibm-cloud