Università di Messina

Dipartimento di scienze matematiche e informatiche, scienze fisiche e della terra

Corso di Laurea Triennale in Informatica

Distributed computation of linear algebra operations

Author:
Gabriele Aloisio *(503264)*

Supervisor:
prof. Roberto Marino

19 settembre 2023

Indice

1 Abstract

2 Introduzione

Matrix operations are fundamental in numerous scientific and computational domains, serving as the building blocks for various applications. However, traditional sequential computation on a single machine often becomes a bottleneck, limiting the speed and scalability of matrix calculations. To address this problem, we are going to use an open-source distributed computing framework called **Ray**.

2.1 Overview of distributed systems and networks

A distributed system consists of multiple interconnected computers that collaborate and coordinate their activities to achieve a common goal. These systems are designed to tackle tasks that cannot be efficiently computed by a single machine. Networks serve as the backbone of distributed systems, enabling communication among the connected nodes. Network infrastructures enable coordination, data sharing, and synchronization across distributed systems, regardless of their physical locations.

2.2 The Ray library

The Ray library is an open-source distributed computing framework primarily designed for building scalable and high-performance applications. It was developed by the company Anyscale, which aimed to simplify the development of distributed and parallel computing applications. When it comes to matrix operation computation, using Ray for distributed computation offers several advantages over performing the operations on a single machine:

- **Faster execution**: With Ray you can use multiple machines and CPUs to compute operations. This significantly reduces the computation time compared to a single machine. Each machine can work on a subset of the matrix data, completing the calculations in parallel. This distributed approach can lead to substantial speedups.
- **Scalability**: As the size of the matrices grows, a single machine may struggle to handle the computational demands due to memory limitations or processing power constraints. Ray allows you to scale horizontally by adding more machines to the distributed setup.
- **Fault tolerance**: Ray offers fault tolerance mechanisms that ensure the continuity of computation even in the presence of failures. If a machine participating in the distributed computation fails, Ray can automatically redistribute the workload to other available machines.
- **Resource utilization**: With Ray, each machine contributes its processing power and memory capacity to the overall computation. This efficient utilization of resources allows you to make the most of the available hardware infrastructure, compared to a single machine that may be underutilized.

3 Caso di studio

4 Implementation

4.1 Python

Python is a versatile and high-level programming language known for its simplicity and readability. It is widely used for a variety of applications, including web development, data analysis, artificial intelligence, scientific computing, and automation. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming, and its large and active community contributes to a vast ecosystem of libraries and frameworks, making it a powerful tool for a wide range of tasks. It is one of the two languages supported by Ray, alongside Java.

4.2 Ray

To parallelize a function with Ray, import Ray and and initialize it with *ray.init()*. Then decorate the function with *@ray.remote* to declare that you want to run this function remotely. Lastly, call the function with *.remote()* instead of calling it normally. This remote call yields a future, a Ray object reference, that you can then fetch with *ray.get*:

```
import ray
ray.init()

@ray.remote
def f(x):
    return x * x

futures = [f.remote(i) for i in range(4)]
print(ray.get(futures)) # [0, 1, 4, 9]

```

4.3 Ray clusters

5 Result of tests

6 Conclusion