UNIVERSITÀ DEGLI STUDI FIRENZE

DIEF
Dipartimento di
Ingegneria Industriale

Bigram/Trigram Generation

Parallel Computing Course

Antonio Castellucci
Michela Crulli

Overview

1. Introduction
    - What are n-grams
    - What we have done
    - Hadoop & Amazon Web Service (AWS)
2. Dataset
3. Implementation
    - Sequential implementation
    - Parallel implementation
    - Distributed implementation
4. Results
    - Test platform
        - Parallel implementation setup
        - AWS
    - Tests
5. Conclusions

Introduction

In this project, we realize a program which generates **bigrams** and **trigrams** with three different approaches:

- a **sequential** implementation (Java)
- a **parallel** implementation (Java threads)
- a **distributed** implementation (Hadoop)

Introduction

- What are n-grams?

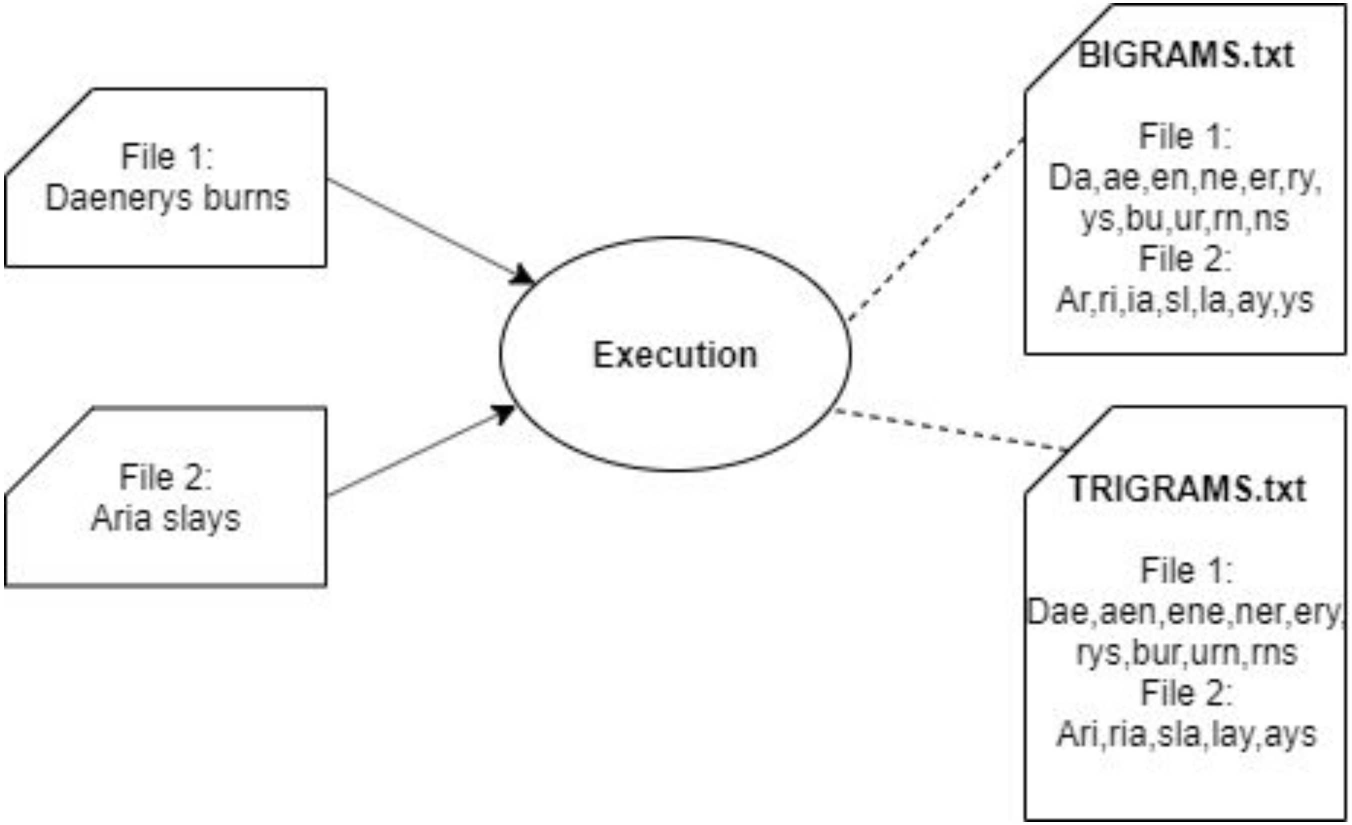
  An n-gram is a **contiguos sequence** of n items (letters, words, syllables or phonemes) from a given text or speech

- Why?

  ○ make new **word predictions**
  ○ make **spelling error corrections**

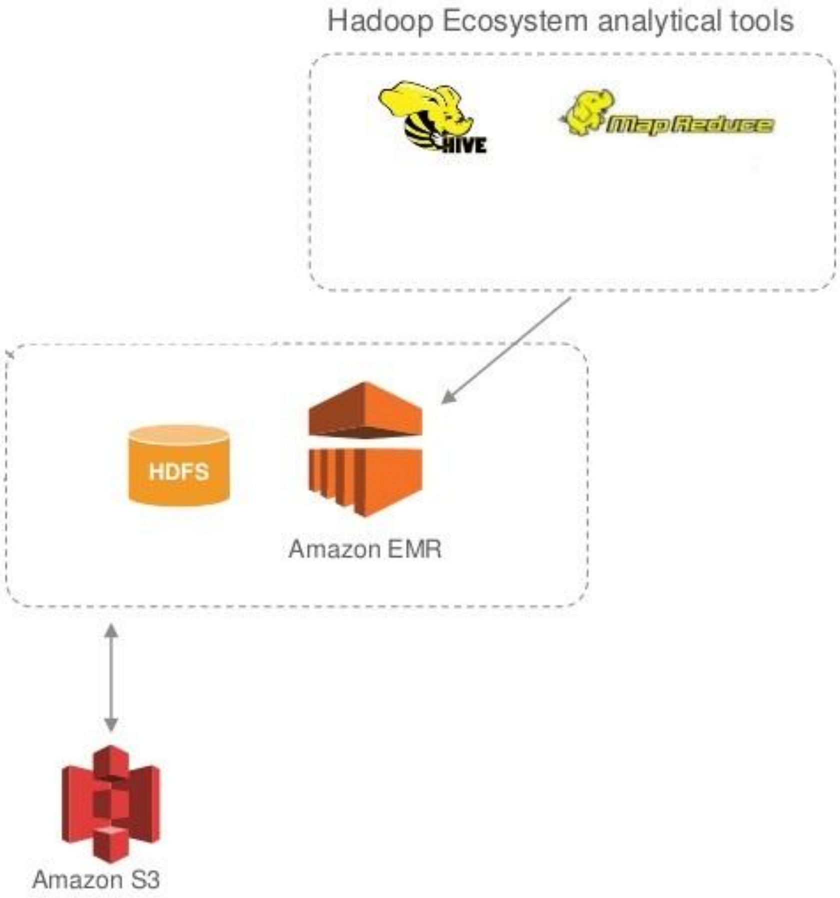
Introduction

- Input output structure



Introduction

- Hadoop & Amazon Web Service (AWS)



amazon web services

Dataset

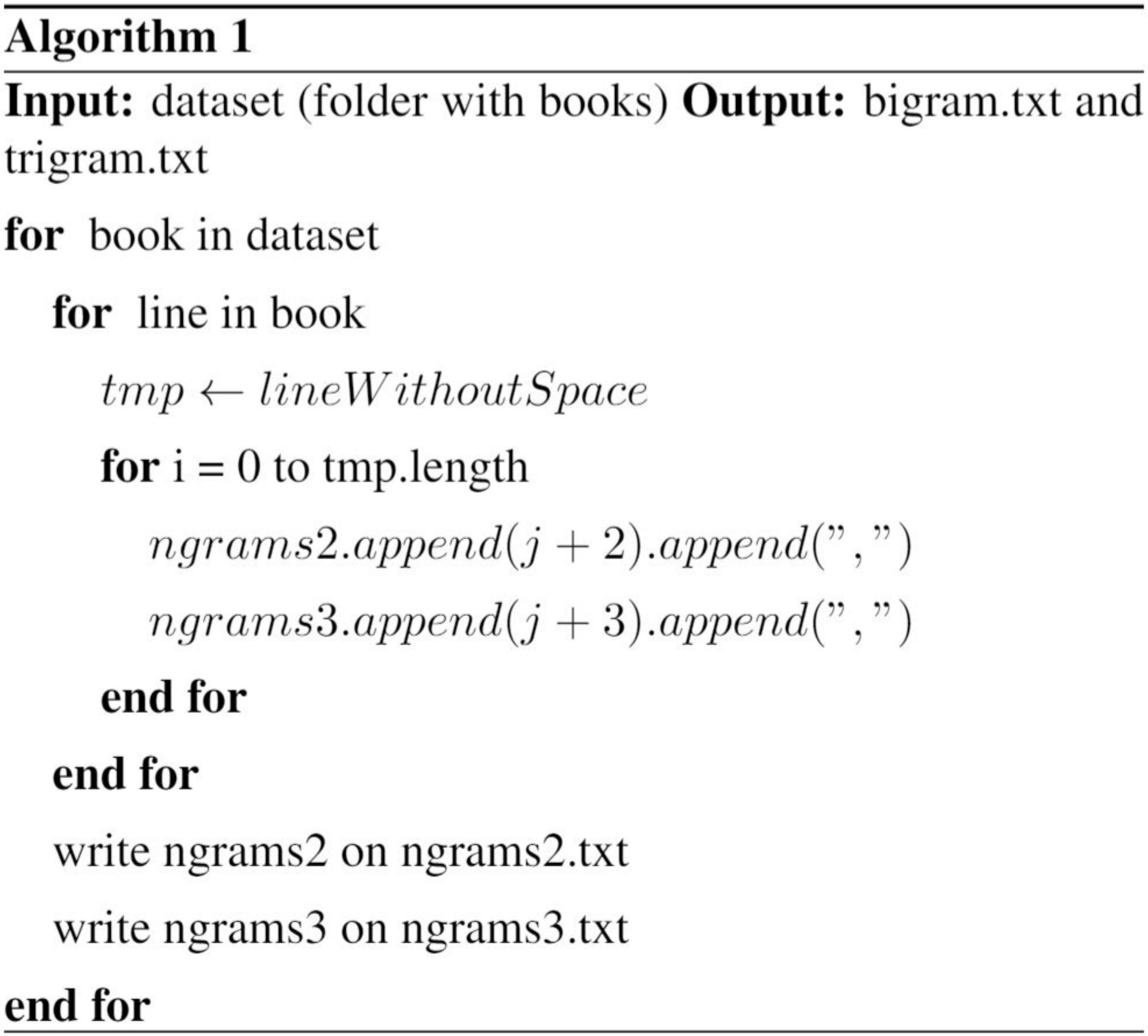
We use:

- a collection of **3000 books** (1.1 GB,from Gutemberg’s Project) and on the collection’s subsets:
  - ○ **2000** (abt 800 megabyte)
  - ○ **1000** (abt 400 megabyte)
  - ○ **500** (abt 80 megabyte)
  - ○ **150** (abt 50 megabyte)
  - ○ **100** (abt 40 megabyte)
  - ○ **50** (abt 20 megabyte)

The size of each book is about **360 KB**

Implementation

- Sequential implementation



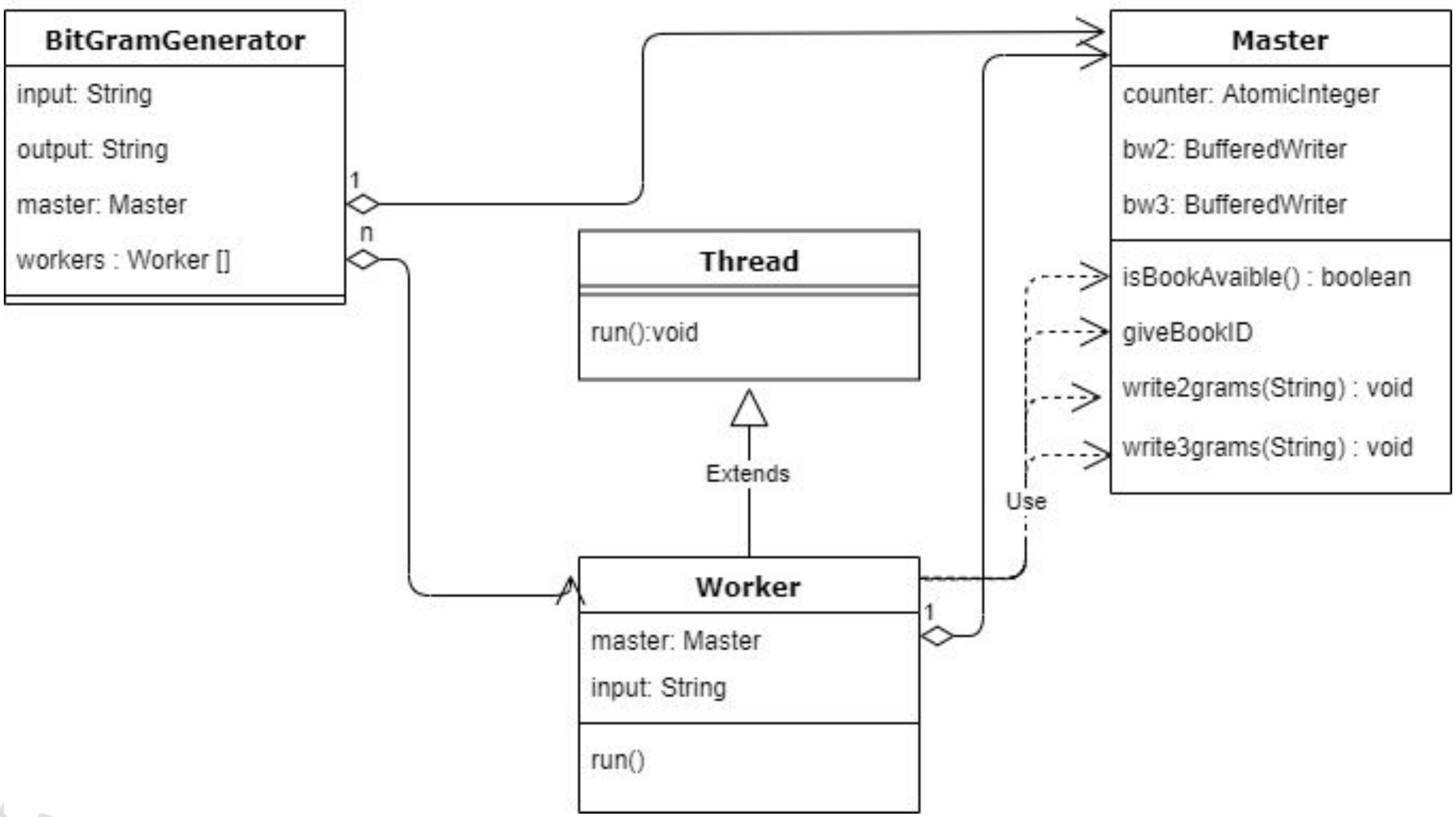
Algorithm 1

Input: dataset (folder with books) **Output:** bigram.txt and trigram.txt

```
for book in dataset
    for line in book
        $tmp \leftarrow lineWithoutSpace$
        for i = 0 to tmp.length
            $ngrams2.append(j+2).append(","))$
            $ngrams3.append(j+3).append(","))$
        end for
    end for
    write ngrams2 on ngrams2.txt
    write ngrams3 on ngrams3.txt
end for
```

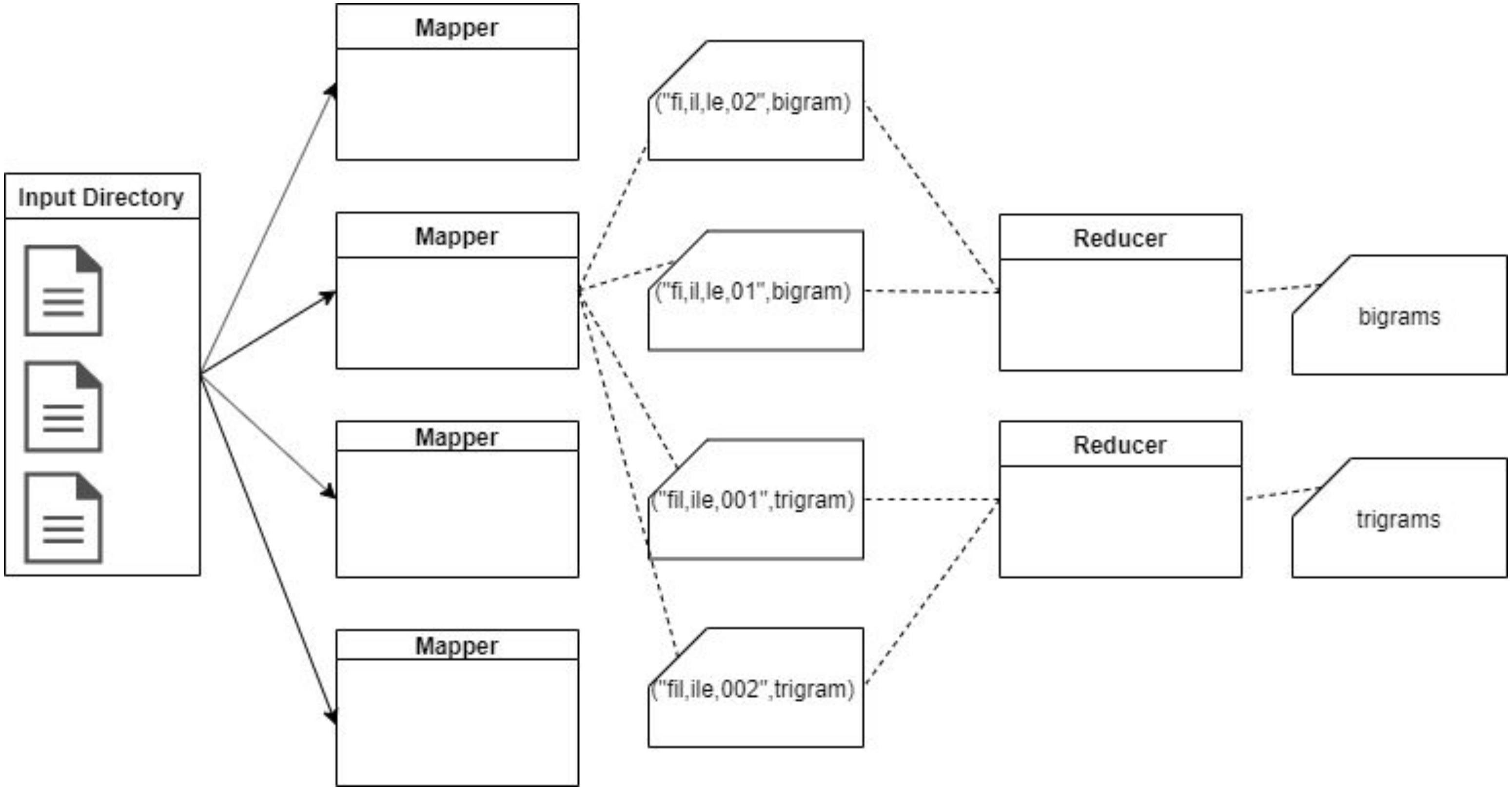
Implementation

- Parallel implementation



Implementation

- Hadoop implementation



Test platform

MSI GS65 with:

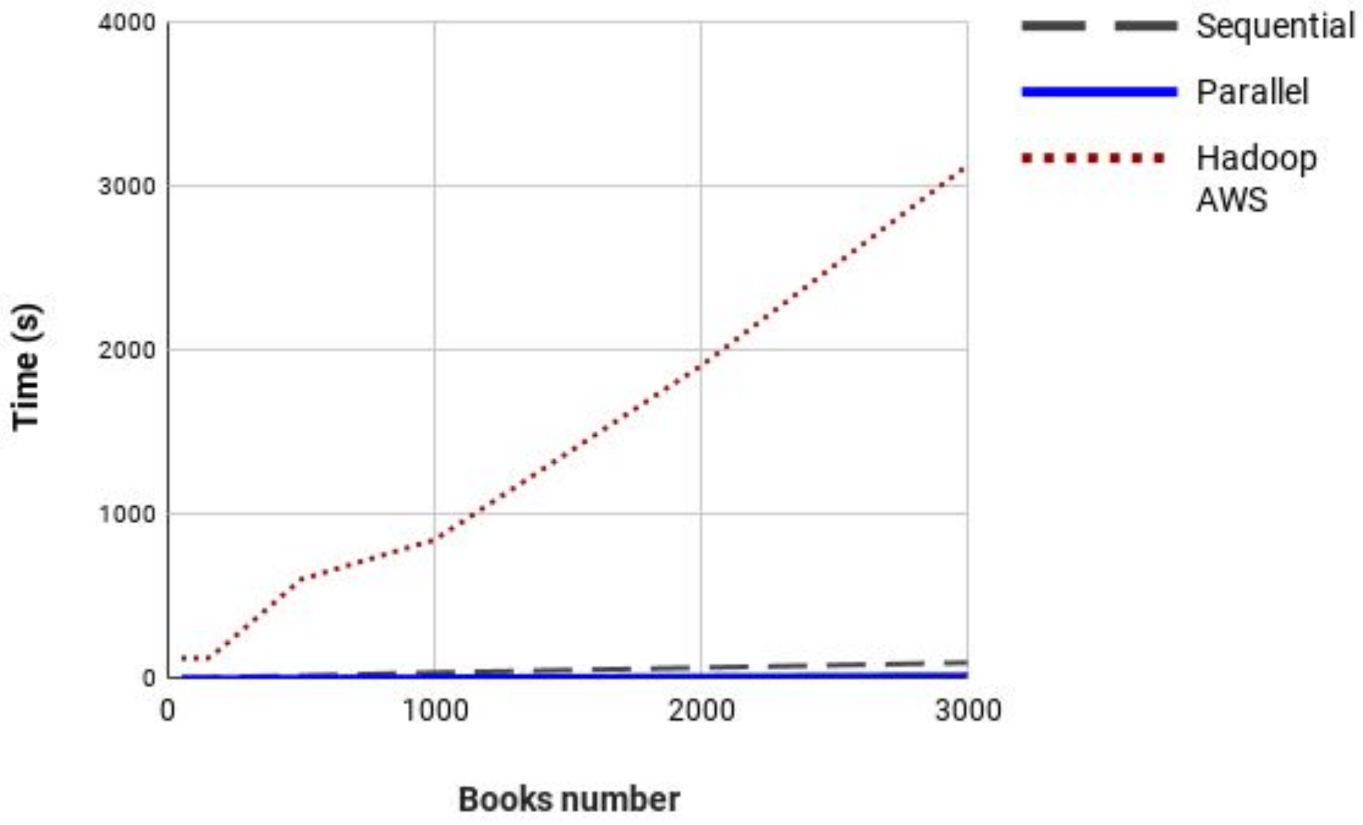
- i7-8750H 6 CORES 12 THREAD
- 16 GB 2400 MHz DDR4 RAM
- M2 SSD SAMSUNG MZVLW256HEHP

Results

- Parellel implementation setup
  - 12 threads and 1 book to process each time
- AWS
  - Emr cluster:
    - m4.xlarge 1 master 4 slaves, Amazon Hadoop version 2.8.5
  - Storage:
    - S3 bucket

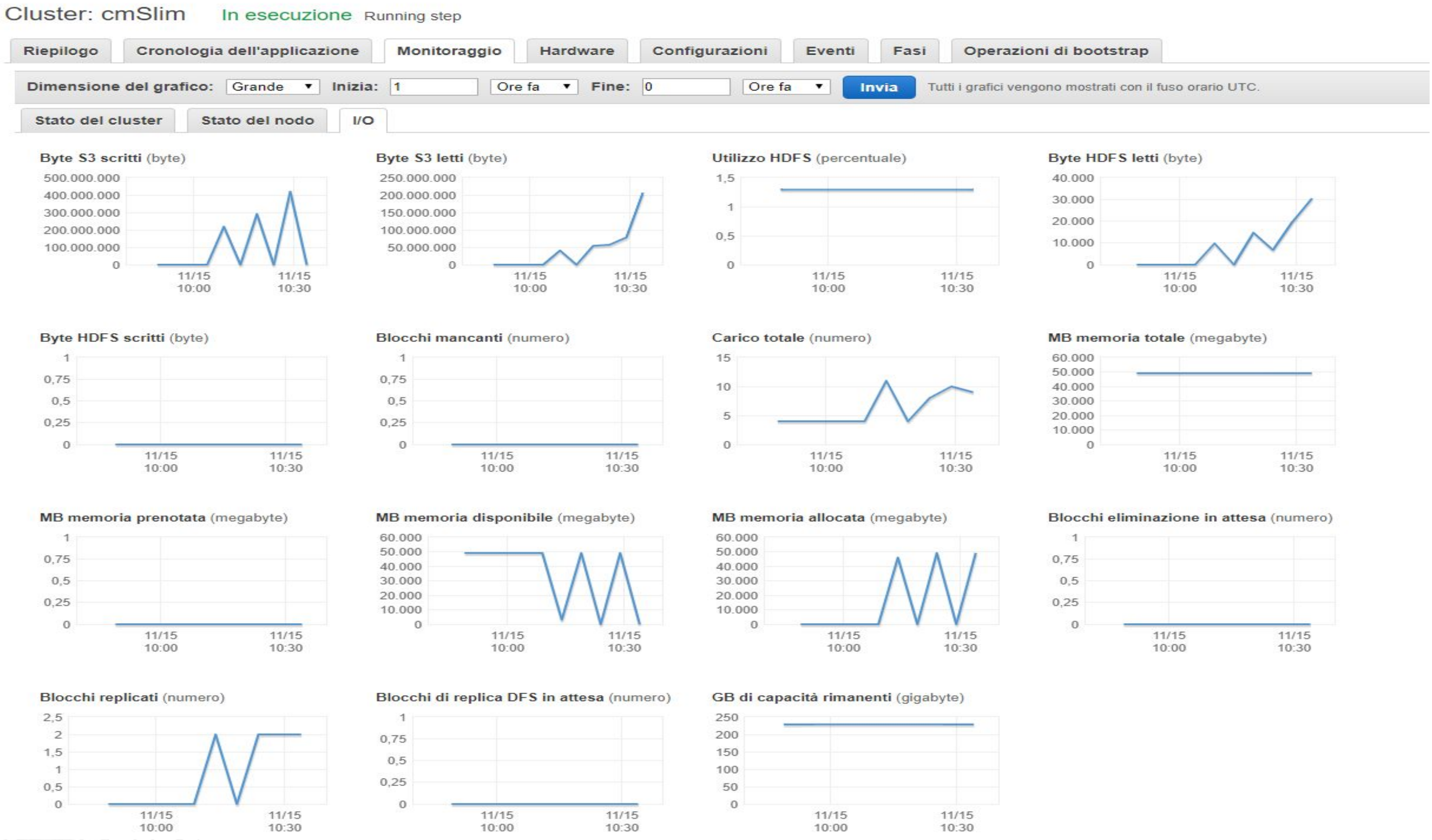
Results

- All versions compared respect to the number of books



Results

- AWS



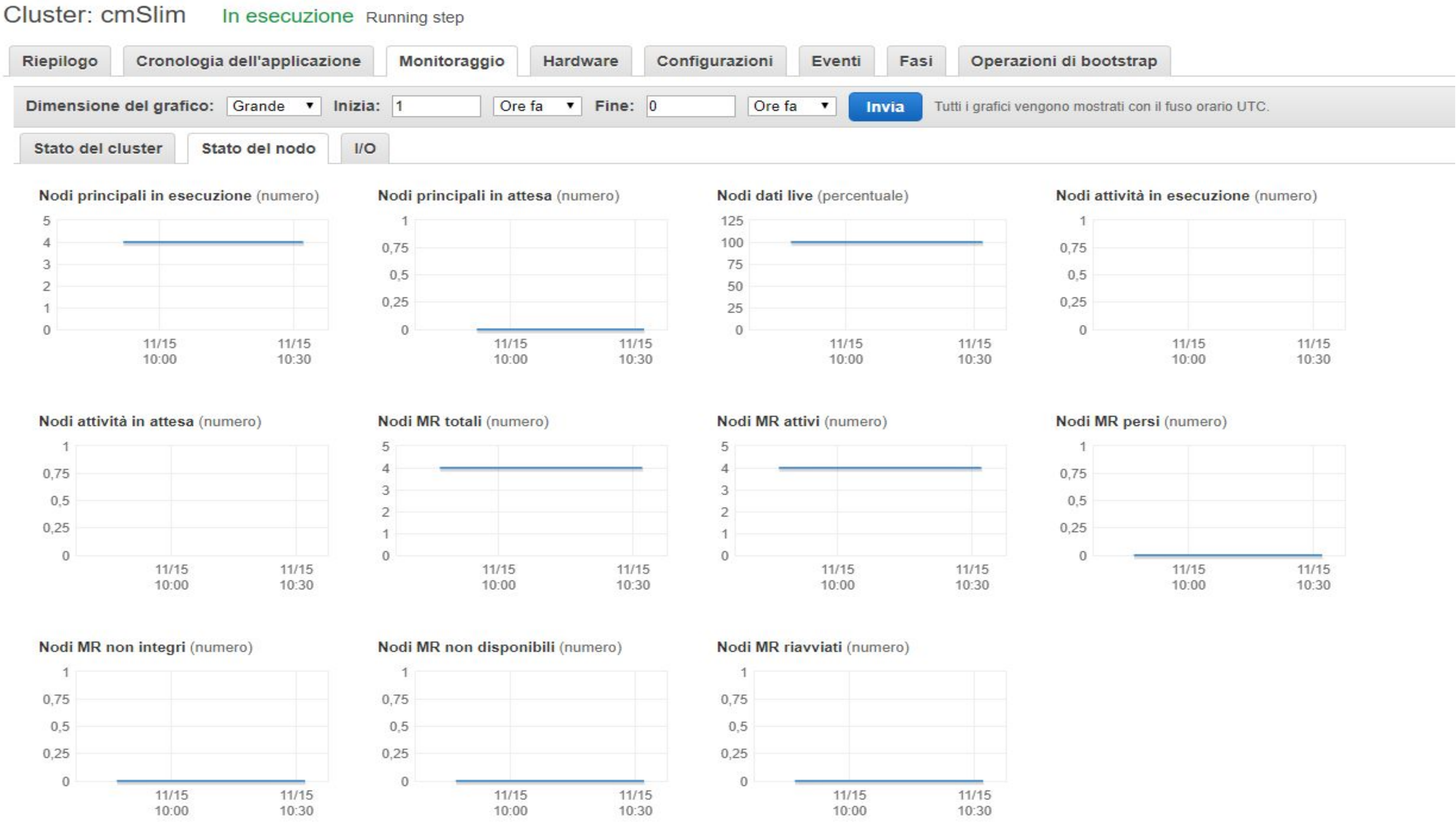
Cluster: cmSlim In esecuzione Running step

Dimensione del grafico: Grande Inizia: 1 Ore fa Fine: 0 Ore fa

Tutti i grafici vengono mostrati con il fuso orario UTC.

Results

- AWS



Cluster: cmSlim In esecuzione Running step

Dimensione del grafico: Grande

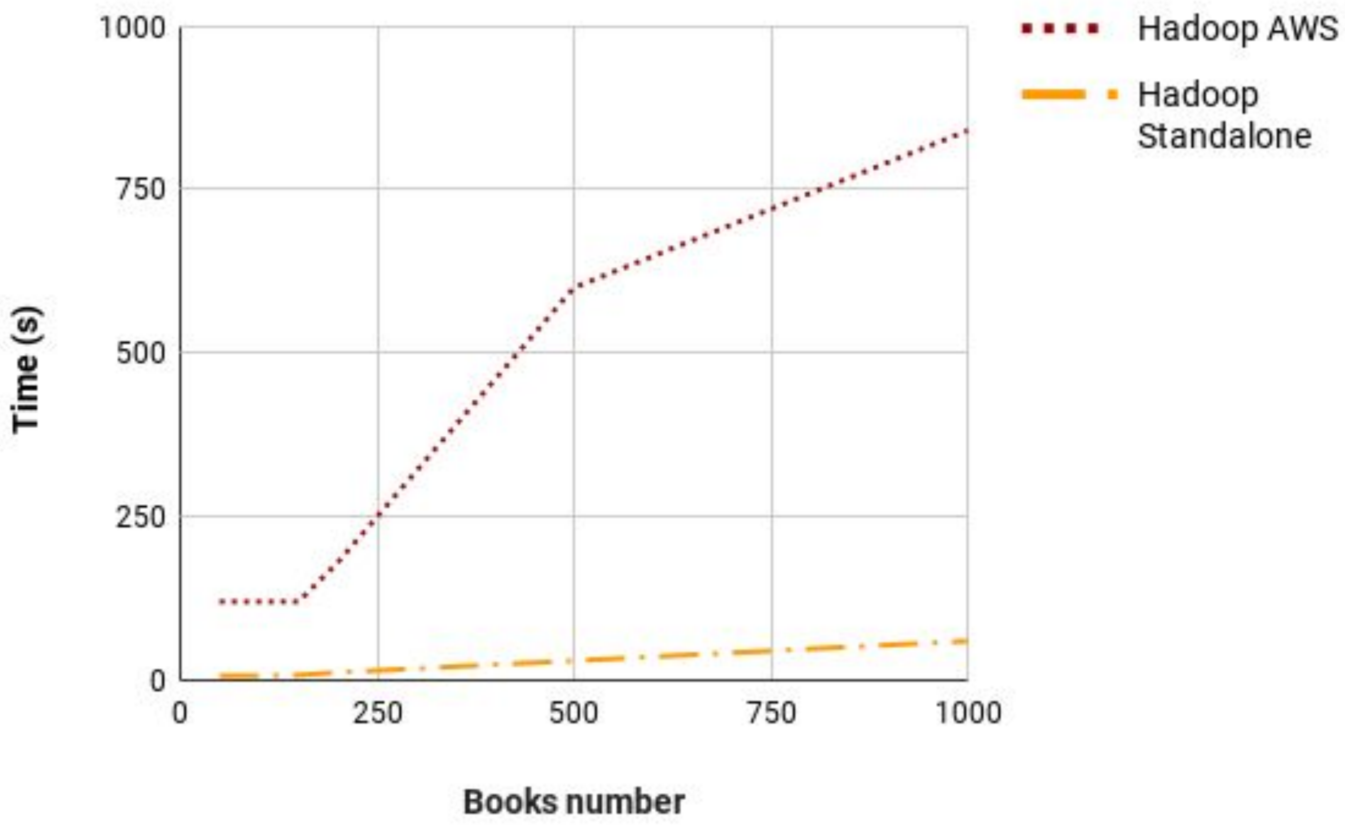
Inizia: 1 Ore fa

Fine: 0 Ore fa

Tutti i grafici vengono mostrati con il fuso orario UTC.

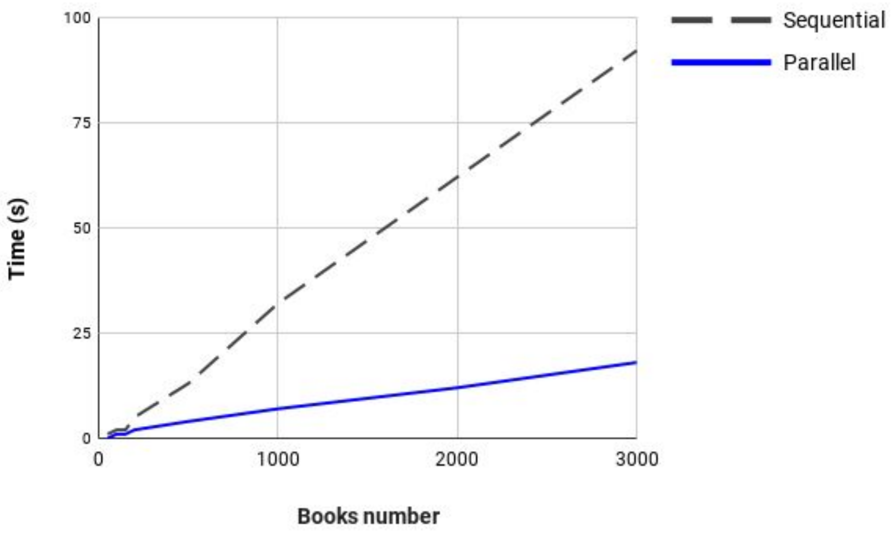
Results

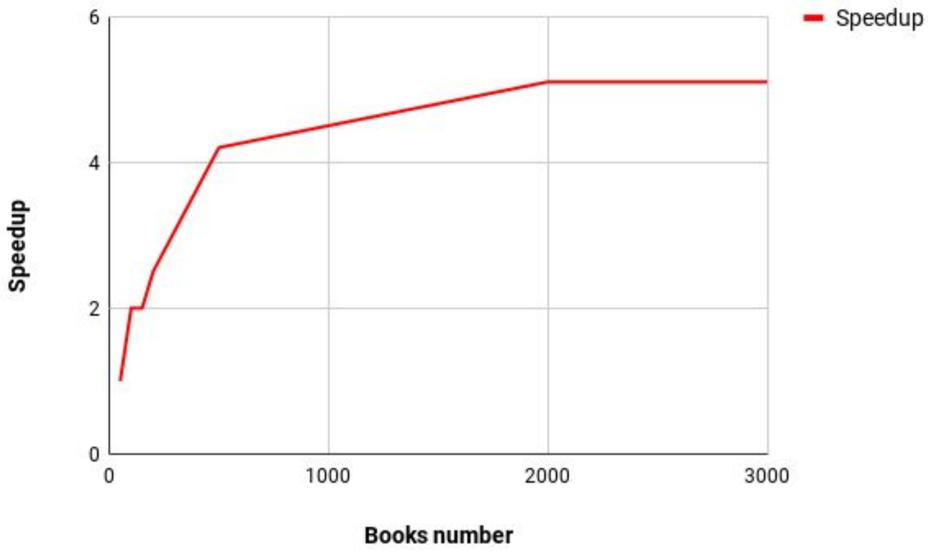
- AWS vs Standalone respect to the number of books



Results

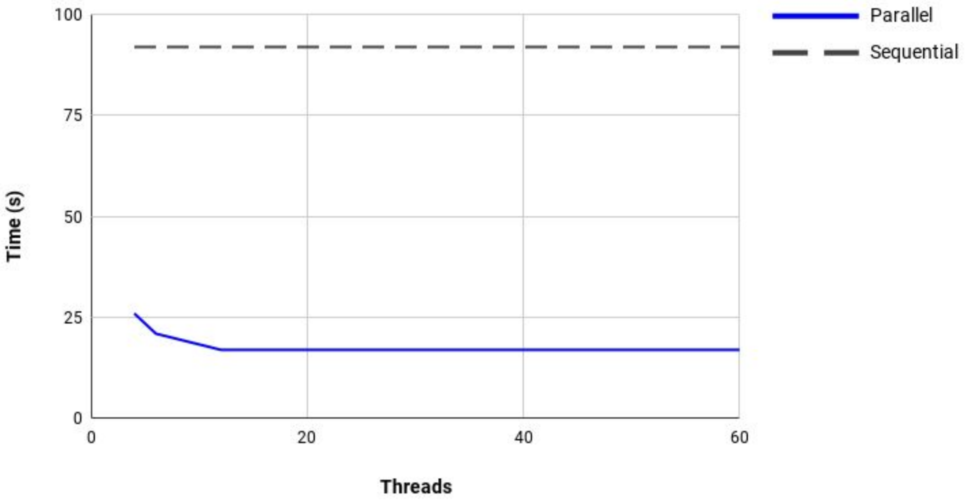
- Parallel vs sequential respect to number of books

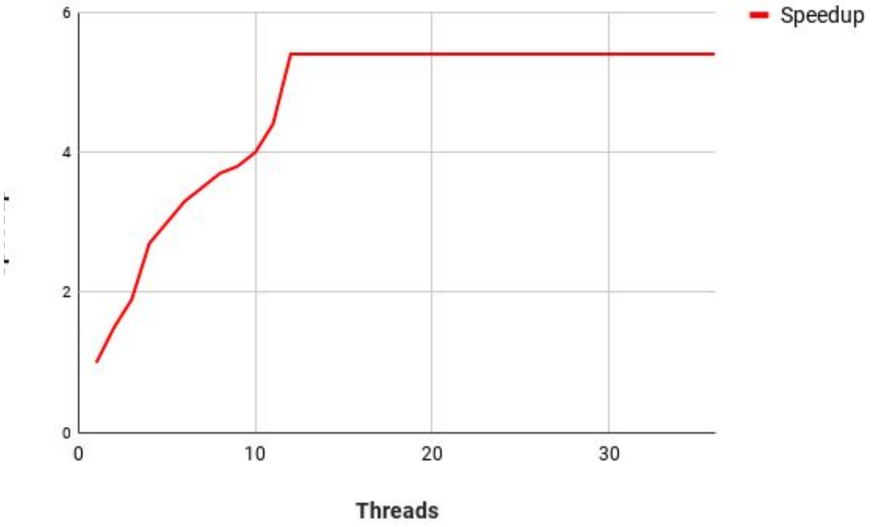




Results

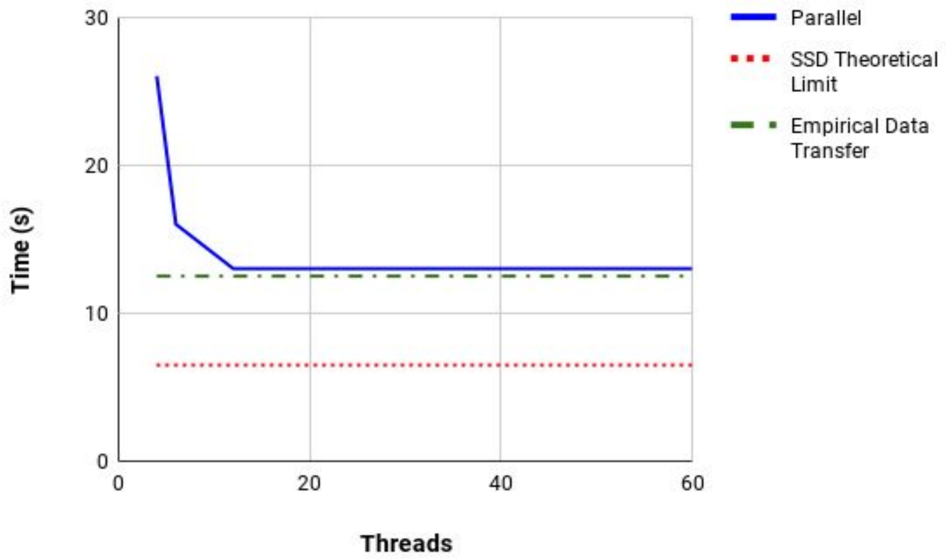
- Parallel vs sequential respect to number of threads

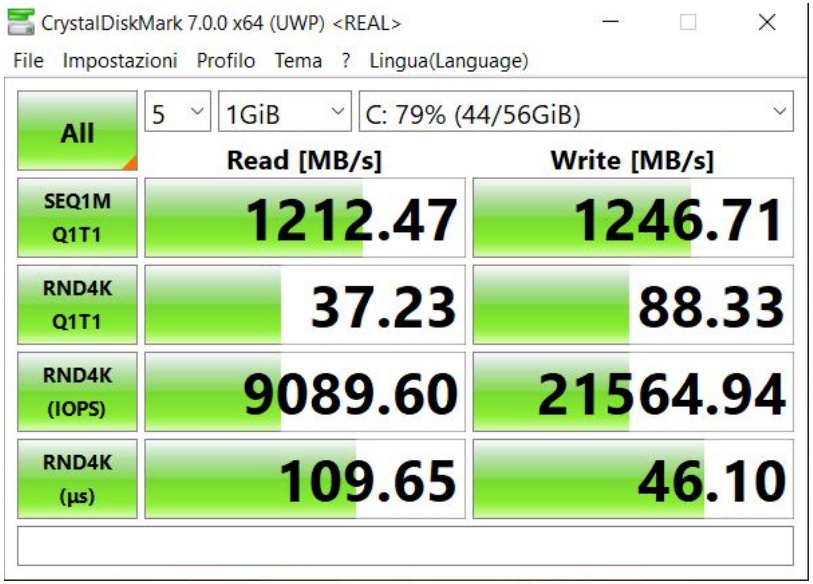




Results

- A deeper analysis





5 | 1GiB | C: 79% (44/56GiB)

All	Read [MB/s]	Write [MB/s]
SEQ1M Q1T1	1212.47	1246.71
RND4K Q1T1	37.23	88.33
RND4K (IOPS)	9089.60	21564.94
RND4K (μs)	109.65	46.10

Conclusions

- Classic case of Hadoop usage:

  ○ perform local computations over huge amounts of input data while returning relatively small result set, which makes our case not suitable to be implemented with Hadoop

- A program for n-grams counting rather than writing them only, probably we would have been able to see the potential of this distributed technology

Conclusions

Thanks for the attention!