MapReduce

May 23, 2023

1 Metody Programowania Równoległego

1.1 Temat: Map Reduce

Wykonał: Paweł Kruczkiewicz

1.2 Algorytm sekwencyjny

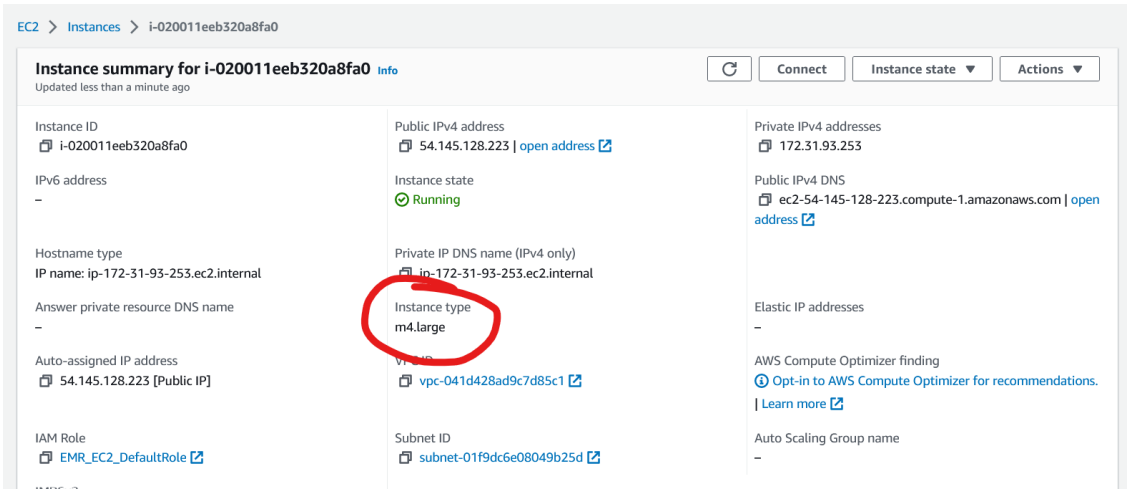
1.2.1 Implementacja

W celu efektywnej implementacji użyto biblioteki `smart_open` stworzonej specjalnie do przetwarzania danych z EC2. Użyty kod:

```
#!/usr/bin/env python
"""word_count_whole_file.py"""

import sys
import time
from smart_open import open

if __name__=="__main__":
    for filename in ["s3://mpr-balis/gutenberg-1G.txt","s3://mpr-balis/gutenberg-5G.txt","s3:/
        for i in range(4):
            word_count = {}
            time1 = time.time()

            for line in open('s3://mpr-balis/gutenberg-1G.txt', encoding="latin-1"):
                words = line.strip().split()
                for word in words:
                    if word not in word_count:
                        word_count[word] = 0
                    word_count[word] += 1

            time_elapsed = round(time.time() - time1, 2)
            print(f"{i+1}\t{filename}\t{time_elapsed}[s]")
```

1.2.2 Konfiguracja

Powyższy kod dla 3 typów wykonano na maszynie EC2 o typie instancji `m4.large`.



Wyniki przedstawiono w dalszej części sprawozdania.

1.3 Algorytm z użyciem paradygmatu *Map Reduce*

1.3.1 Implementacja

Użyto standardowej implementacji `mapper.py`

```
#!/usr/bin/env python
"""mapper.py"""

import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        print '%s\t%s' % (word, 1)
```

oraz `reducer.py`

```
#!/usr/bin/env python
"""reducer.py"""

from operator import itemgetter
```

```
import sys

current_word = None
current_count = 0
word = None

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # write result to STDOUT
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word

# do not forget to output the last word if needed!
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

1.3.2 Konfiguracja

Powyższy kod zastosowano w paradygmacie `Map Reduce` na maszynie `EMR` w `Amazon Cloud Service`. Dokładne konfiguracje zostały opisane niżej.

Aby uruchomić i sprawdzić powyższy kod *dla każdego klastra*: 1. Przesłano plik `gutenberg-1g.txt` za pomocą `scp`. 2. Rozpakowano go i wrzucono na hadoopa za pomocą komend `hdfs dfs -touchz` oraz `hdfs dfs -appendFileTo`. 3. Stworzono pliki `mapper.py` oraz `reducer.py` z treścią jak wyżej. 4. Zamieszczono i użyto skryptu `script.sh` przechwytującego `time` do pliku `time.txt`. 5. Przeanalizowano plik `time.txt` i dodano odpowiednie linie do pliku `result.csv` zawierającego

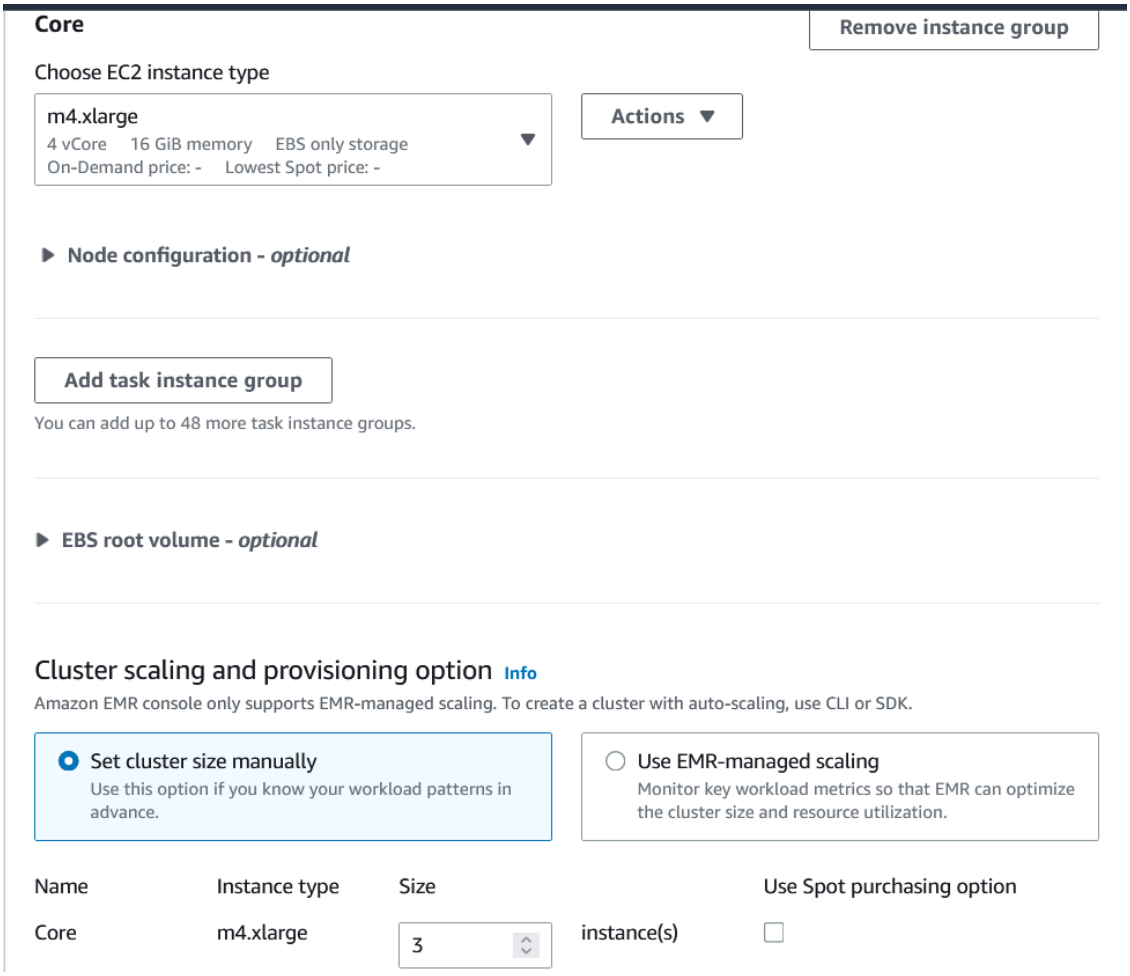
wyniki eksperymentu w formie pliku `csv`.

Plik `script.sh`

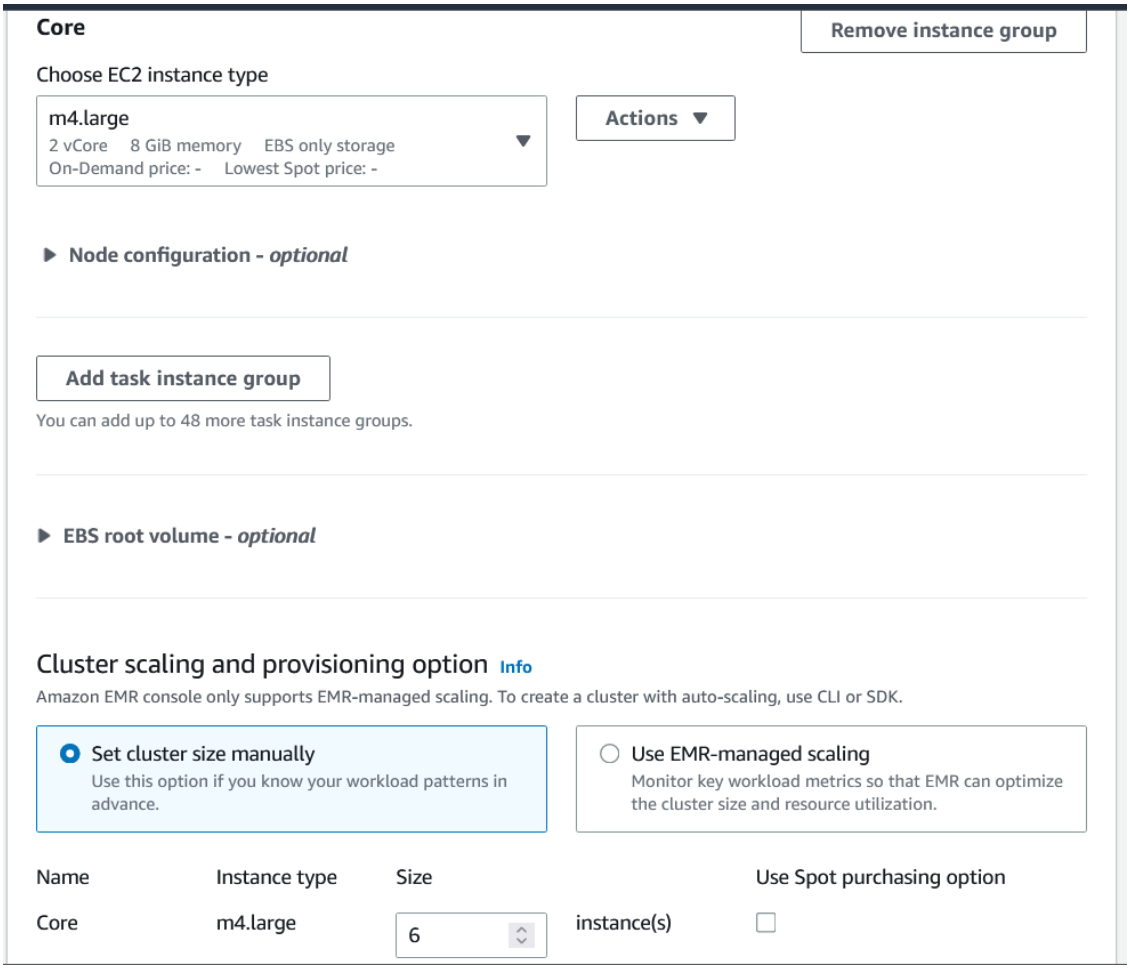
```
for size in 1 5 10
do
        input="books-input/${size}g.txt"
        for test_case_num in {1..3}
        do
                { time hadoop jar /usr/lib/hadoop/hadoop-streaming.jar \
                                -files mapper.py,reducer.py\
                                 -mapper mapper.py -reducer reducer.py \
                                -input ${input} -output books-output ; } 2>>time.txt
                hdfs dfs -rm -r books-output
        done
done
```

Wielkość konfiguracji to użyte sumarycznie max 12 core'ów.

Pierwsza konfiguracja Pierwsza konfiguracja to **3 x 4 core'y** (3 instancje typu `m4.xlarge`).



Druga konfiguracja Druga konfiguracja to **6 x 2 core'y** (6 instancji typu `m4.large`).



1.4 Wyniki

1.4.1 Plik CSV

Dane z poszczególnych plików time przeanalizowano i dodano do pliku `result.csv`.

[28]:
```
import pandas as pd
import matplotlib.pyplot as plt

results = pd.read_csv("result.csv")
results.head()
```

[28]:

	nCores	confId	dataSize	time
0	1	0	1	90.68
1	1	0	1	89.56
2	1	0	1	89.83
3	1	0	1	89.96
4	1	0	5	466.19

Dane następnie zgrupowano wg typu konfiguracji (`confId`) oraz wielkości danych (`time`).

[35]:
```
grouped_res = results.groupby(['nCores', 'dataSize'])
agg_res = grouped_res.agg(["mean", "std"])

agg_res
```

[35]:

		confId		time	
		mean	std	mean	std
nCores	dataSize				
1	1	0.0	0.0	90.007500	0.478287
	5	0.0	0.0	467.306667	1.664702
	10	0.0	0.0	953.816667	9.075992
3	1	1.0	0.0	227.116667	2.090606
	5	1.0	0.0	919.550000	36.918509
	10	1.0	0.0	1771.346667	20.201139
6	1	2.0	0.0	261.946667	66.417748
	5	2.0	0.0	1010.610000	51.736485
	10	2.0	0.0	2103.106667	169.922218

[67]:
```
base_time = results[results["nCores"] == 1].groupby("dataSize").agg(["mean",␣
 ↪"std"]).transform(lambda x: x["time"])
base_time
```

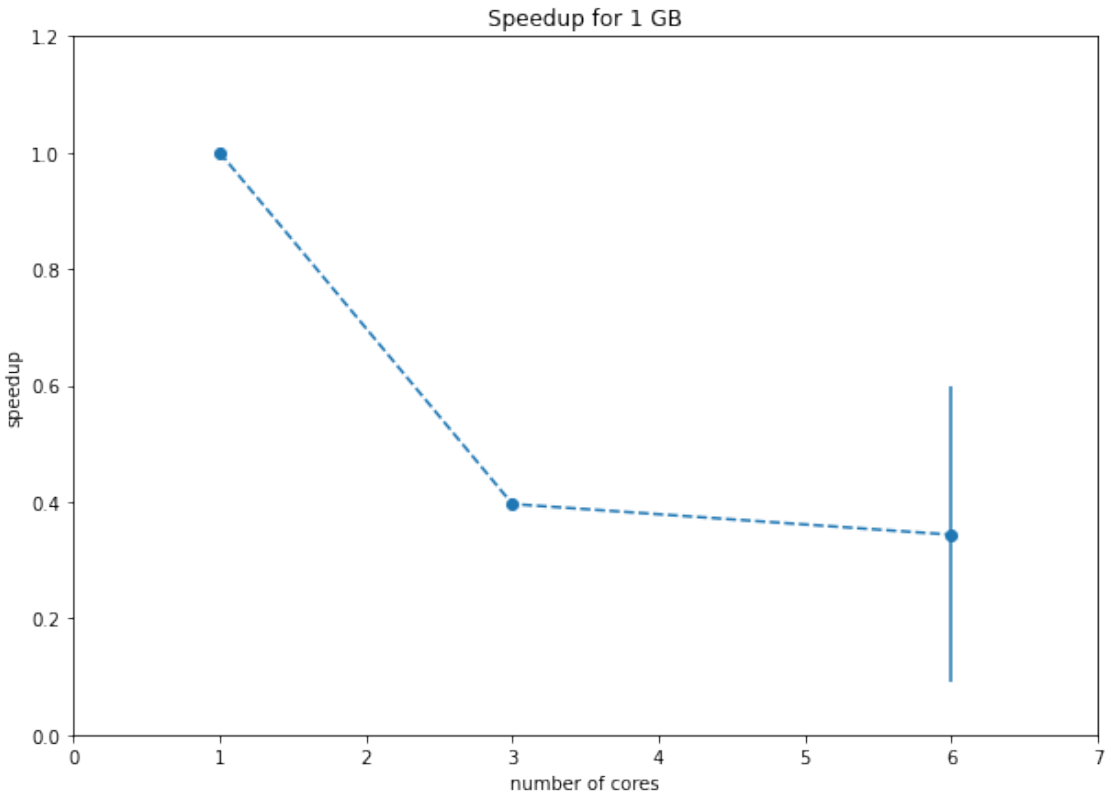
[67]:

	mean	std
dataSize		
1	90.007500	0.478287
5	467.306667	1.664702
10	953.816667	9.075992

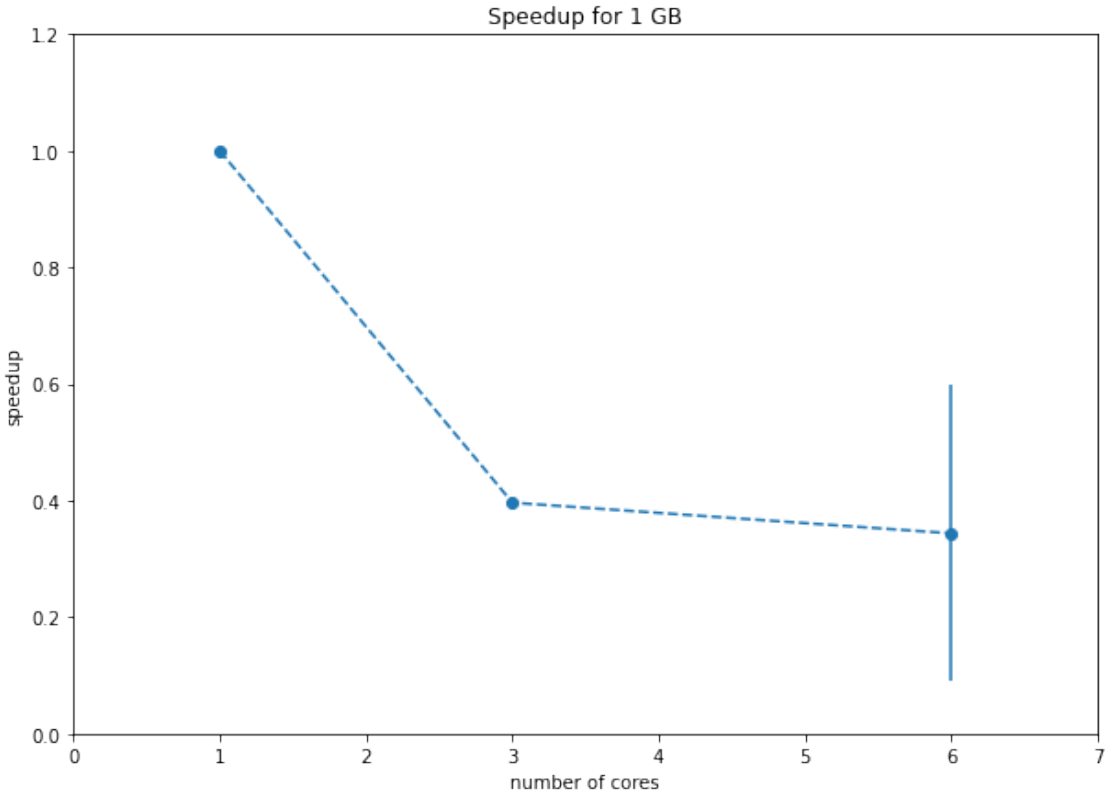
Następnie policzono speedup w zależności od użytych core'ów dla wszystkich trzech konfiguracji. Jako wartość bazowa (w mianowniku) posłużył *czas wykonania algorytmu sekwencyjnego na ec2*

[76]:
```
#wyiczenie speedupu
def get_speedup(dataSize):
    df = results[results["dataSize"] == dataSize].groupby("nCores").
 ↪agg(["mean", "std"])
    time_mean = df["time"]["mean"]
    speedup_mean = base_time["mean"][dataSize] / time_mean
    speedup_std = df["time"]["std"] / time_mean
    return speedup_mean, speedup_std

def plot_line_with_errors(x, y, e, title, x_label, y_label, ymax,␣
 ↪diagonal=False, hline=None):
    f, ax = plt.subplots(1)
    f.set_size_inches(10,7)

    ax.errorbar(x, y, e, marker="o", linestyle="--")
```

```
    if diagonal: ax.plot(x, x, linestyle="--")
    if hline: ax.axhline(hline, linestyle="--", color="red")

    plt.title(title)
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    ax.set_xlim(xmin=0, xmax=7)
    ax.set_ylim(ymin=0, ymax=ymax)

    plt.show(f)


x_axis = [1, 3, 6]
```



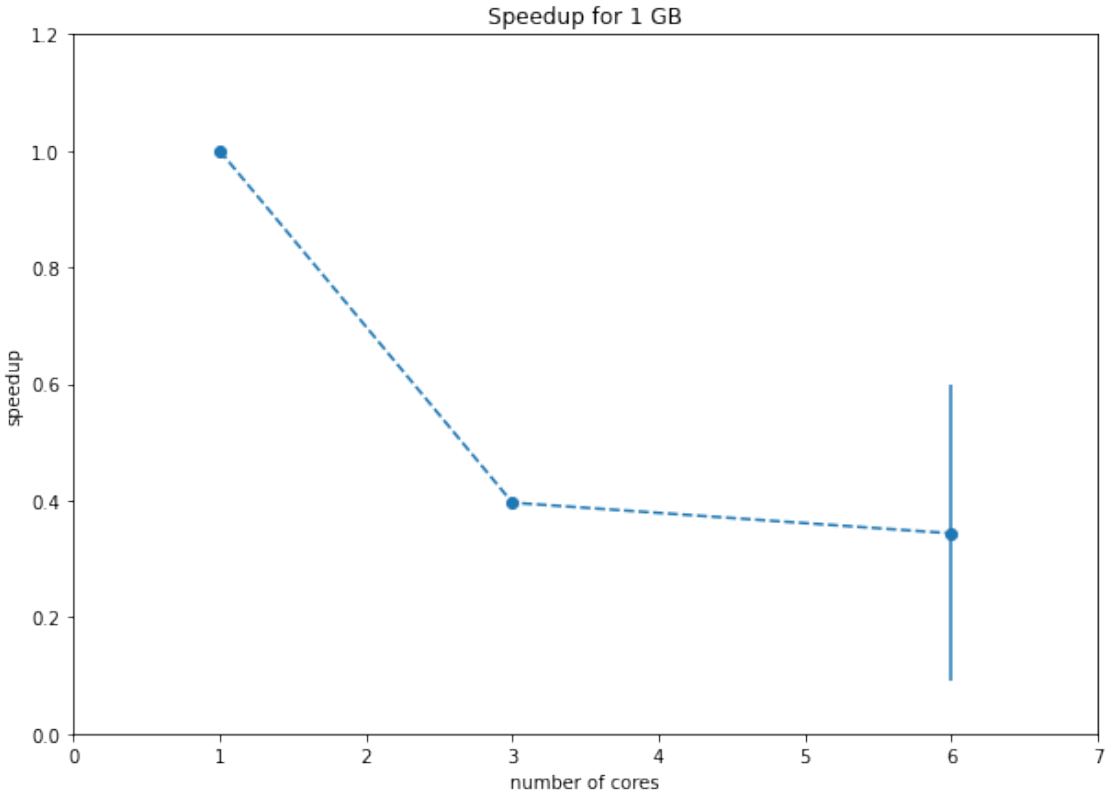
[77]:
```
# 1G
```



1.4.2 Wykresy

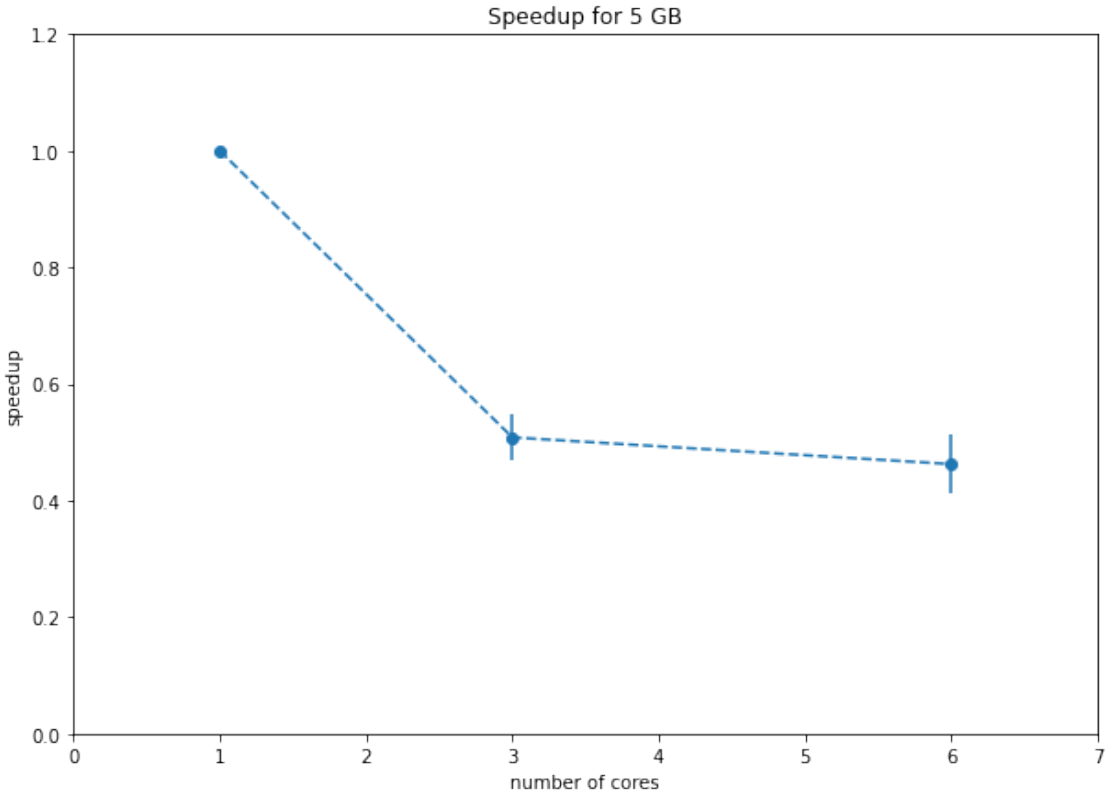
1 Gb

[78]:
```
speedup_1, speedup_1_std = get_speedup(1)
plot_line_with_errors(x_axis, speedup_1, speedup_1_std, "Speedup for 1 GB",␣
↪"number of cores", "speedup", 1.2, diagonal=False)
```



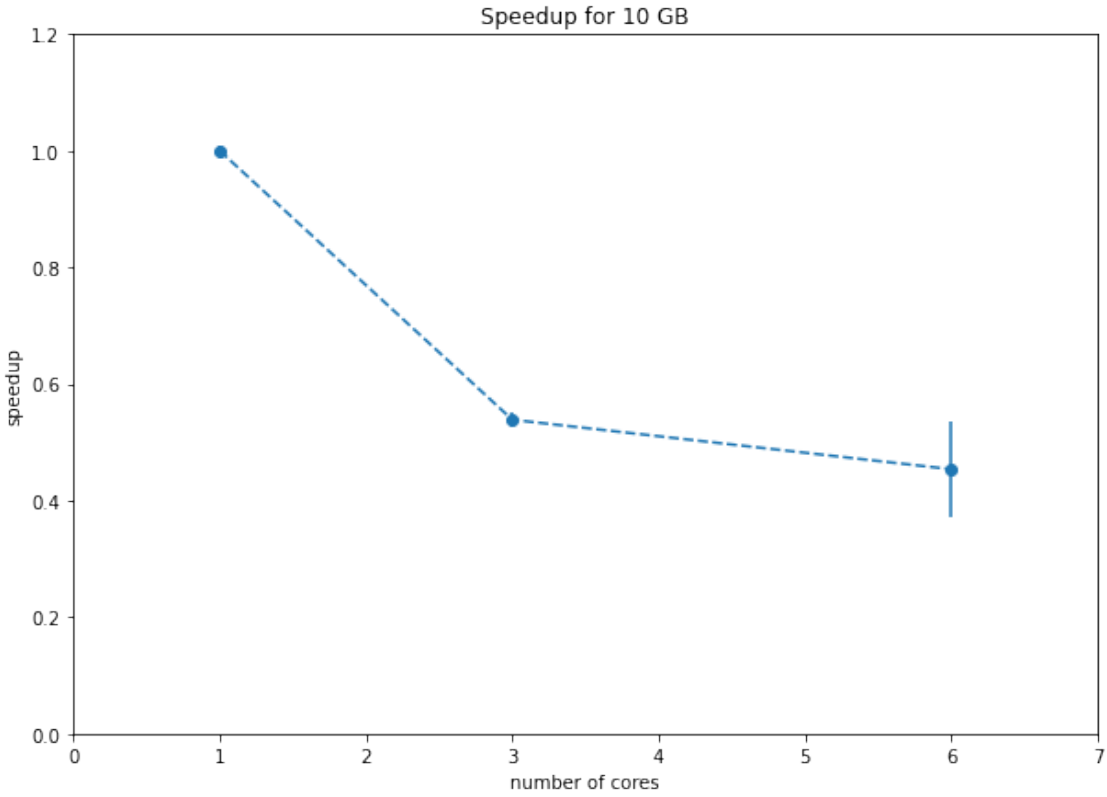
wykres dla 5 GB

[82]:
```
speedup_5, speedup_5_std = get_speedup(5)
plot_line_with_errors(x_axis, speedup_5, speedup_5_std, "Speedup for 5 GB",␣
↪"number of cores", "speedup", 1.2, diagonal=False)
```



wykres dla 10 GB

[81]:
```
speedup_10, speedup_10_std = get_speedup(10)
plot_line_with_errors(x_axis, speedup_10, speedup_10_std, "Speedup for 10 GB",␣
↪"number of cores", "speedup", 1.2, diagonal=False)
```



1.4.3 Komentarz

Przedstawione wyżej wykresy wyraźnie pokazują, że żadna konfiguracja nie jest w stanie poprawić wyniku z EC2. Bazując na pracy *Scalability! But at what COST?* możemy hipotetyzować, że jest to spowodowane dodatkowym narzutem spowodowanym na rozdsponowanie plików po wielu komputerach. Na bazie tendencji spadkowej wykresów przyśpieszenia możemy stwierdzić, że **badana implementacja paradygmatu mapreduce ma *bezgraniczną* (ang. *unbounded*) metrykę COST**. Oznacza to, że prawdopodobnie nie istnieje konfiguracja, na której rozwiązanie problemu nastąpi szybciej niż na jednowątkowym komputerze z optymalną implementacją rozwiązania.

Warto jednak zauważyć dwie rzeczy:

1. Hadoopowe przetwarzanie map reduce działało nieco lepiej dla plików o rozmiarze 5 i 10 GB niż dla 1 GB (Speedup ok. 0.5 zamiast 0.37). Pliki są mniejsze niż te, dla których Hadoop został stworzony.
2. Konfiguracja z 6 core'ami jest słabsza ze względu na ograniczenia AWSa. Być może użycie wyższego typu sprzętu pozwoliłoby *dogonić* implementację na jeden wątek.