1. FEEDBACK

Partile cele mai grele pe care le-am intampinat si au necesitat mai mult timp de prelucrare in timpul rezolvarii temei au fost: crearea arhivei cu specificatiile mentionate si rezolvarea problemelor de sincronizare, atunci cand incerc sa adaug toate listele procesate in listele globale.

2. STRATEGIA DE PARALELIZARE

Pentru paralelizarea programului, fiecare thread trebuie sa se ocupe de urmatoarele task-uri:

1. Citirea fisierelor din lista de JSON-uri, maparea continutului unui astfel de fisier intr-o lista de articole, si adaugarea articolelor in 3 liste: lista locala fiecarui thread care contine toate articolele procesate de catre acesta si in listele globale, statice ale clasei MyThread, seen_uuids si seen_titles, in care sunt stocate toate aparitiile unui uuid, specific unui titlu

2. In a doua parte, ma folosesc de o bariera pentru a fi sigur ca toate thread-urile au procesat toate fisierele. Dupa, fiecare thread parcurge lista de articole locale si verifica: unicitatea unui articol (daca numarul de aparitii al uuid-ului si al titlului este egal cu 1). Daca este un articol unicat, putem sa-l procesam mai departe si sa adaugam continutul acestuia in urmatoarele liste locale: categories_list, languages_list, keywords_list, most_recent_articles, authors_list.

3. Apoi thread-urile isi incarca listele procesate anterior in liste globale, sincronizate, accesate prin instanta “instance” de tip Statistics.

4. In ultima parte, paralelizarea se opreste: numai thread-ul cu id-ul 0 se ocupa de printarea in fisiere al datelor stocate in “instance” (partea de printare se ocupa si de sortarea continutului listelor).

Mecanismele de sincronizare folosite in aceasta tema sunt:

1. Metode de tip synchronized, pentru a bloca accesul multiplelor thread-uri de a adauga elemente in liste.
2. Structura ConcurrentHashMap pentru a bloca accesul thread-urilor la anumite elemente, la un moment dat.
3. AtomicInteger pentru a gestiona accesul la o variabila intreaga, pentru a nu folosi tot timpul metoda synchronized.

Implementarea mea este corecta fiindca ma folosesc de conceptul de paralelizare: folosesc fire de executie care separa lista de articole originala in liste de dimensiuni mai

mici, care pot fi procesate mult mai usor si mai rapid. Procesarea presupune: citirea articolelor, verificarea unicitatii acestora si crearea listelor pentru: limbi, categorii, cele mai recente articole publicate, cele mai folosite cuvinte si autori.

3. ANALIZA DE PERFORMANTA SI SCALABILITATE

1. SETUP DE TESTARE:

Versiune Java folosita: openjdk 25.0.1

Dimensiune dataset de test: 13,789 de fisiere de tip .JSON

Configuratia sistemului: Linux, Ubuntu 24.04.3 LTS, 12th Gen Intel Core i7-12700H Processor, 16 GB RAM, 14 core-uri de procesare

2. REZULTATE:

Ca set de date vom folosi articolele specificate de la testul 5, adica 13,789 de fisiere .JSON, care pot contine intre 1 si 10 articole per fisier. Mai jos am atasat un tabel cu urmatoarele date: 1 linie: Ti (unde i este numarul de thread-uri folosite), urmatoarele 3 linii timpi pentrecut de i procese sa termine de executat testul, iar ultima linie este timpul mediu pentru executia codului.

T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
17.825	10.379	7.988	6.721	6.034	5.514	5.78	5.547	5.277	5.004	4.898
17.793	10.747	7.894	6.793	5.818	5.612	5.757	5.438	5.236	5.109	4.897
18.294	10.751	8.09	6.718	6.164	5.634	5.712	5.465	5.233	5.234	5.034
17.970	10.625	7.990	6.744	6.005	5.587	5.75	5.483	5.249	5.116	4.943

T12
4.729
4.907
4.866
4.834

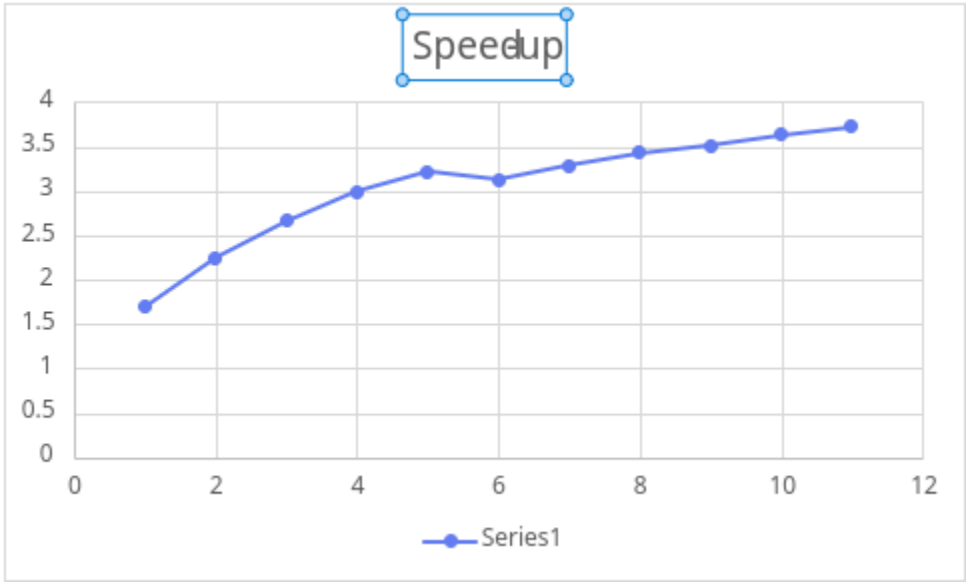
Urmatorul tabel contine speed-up-ul calculat folosind formula $S(P) = \frac{T(1)}{T(P)}$, unde T(1) si T(i) sunt valoarile medii ale timpului pentru executia programului folosind i thread-uri.

S(2)	S(3)	S(4)	S(5)	S(6)	S(7)	S(8)	S(9)	S(10)	S(11)	S(12)
1.69	2.24	2.66	2.99	3.21	3.12	3.27	3.42	3.51	3.63	3.71

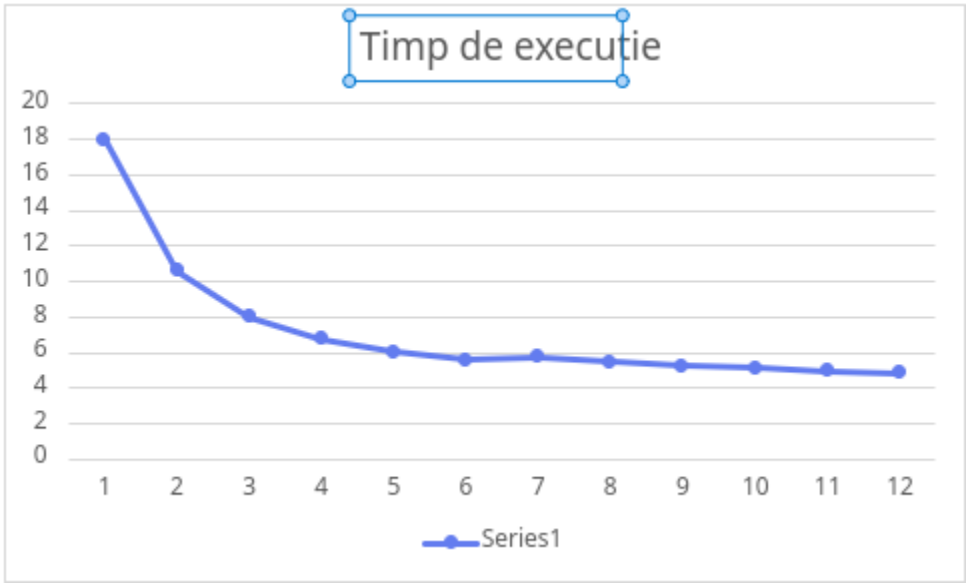
Urmatorul tabel contine eficienta calculata folosind formula $E(P) = \frac{S(P)}{P}$, unde P reprezinta numarul de thread-uri.

E(2)	E(3)	E(4)	E(5)	E(6)	E(7)	E(8)	E(9)	E(10)	E(11)	E(12)
0.84	0.74	0.66	0.6	0.53	0.44	0.41	0.38	0.35	0.33	0.31

Grafic pentru Speed-up:



Grafic pentru timpul de executie:



Observatii:

Performanta creste din momentul in care incep sa folosesc 2 sau mai multe fire de executie, iar timpul de executie incepe sa se stabilizeze de la 6 thread-uri in sus (continua sa devina mai rapida executia programului, dar cu mici decrementari, aproape infim).

Cauze posibile ale limitarii executiei programului: overhead pentru adaugare si sincronizarea datelor in listele globale, dimensinea dataset-ului, deschiderea, scrierea si inchiderea fisierelor de iesire si sortarea elementelor.

Numarul optim de procesoare pentru sistemul meu ar fi 5 (nu trebuie sa folosesc mai mult de 5 fire de executie pentru a obtine o performanta ridicata si pentru a nu folosi toate core-urile sistemului). De la 6 thread-uri in sus, executia programului nu se imbunatateste in mod vizibil.