Feedback
Mi-a placut tema asta din cauza ca mi se pare foarte practica, poate chiar un schelete bun pentru viitoare proiecte, am cateva idei bune in minte.
Implementarea a durat de cand am trimis tema la ISC pana azi. Nu o sa stau sa vad cat, dar destul, nu am fost la cursuri.
Checkerul mi se pare suspect, imi da intre 66 si 75 de puncte pentru scalabilitate.
Nu cred ca mi-a dat niciota o distributie consistenta de timpi la test.
Strategie de paralelizare
Am folosit citire paralela, filtrare secventionala, procesare de cuvinte cheie paralela si scriere secventiala.
Am folosit LinkedList cu pathurile pentru .json-uri asa pentru a paraleliza scoaterea de informatii. Functioneaza mai bine pentru un volum din ce in ce mai mare de dataseturi.
Pentru a scoate duplicatele, am facut un map, si daca un uuid sau unn titlu apare de mai mult decat o data, se scot toate aparitiile.
Pentru a procesa Keyword-urile am folosit acelasi lucru ca mai sus.
Au fost folosite metode sincronizate, blocuri sincronizate, hashmap concurent si thread.join pentru a separa etapele.
Analiza
SPECS:
16GB RAM
Architecture: x86_64
CPU(s): 16
Model name: 12th Gen Intel(R) Core(TM) i5-12500H
Pentru rezultate o sa rulez de cateva ori checkerul si o sa pun cel mai mic si cel mai mare rezultat al scriptului. Sper ca imi ajuta cauza daca zic ca am avut si 75/75 de cateva ori.
Dupa 3 rulari am avut, chiar am avut 75 o data. O sa atasez si un video cu dovada.
Speedup si Eficienta
Am folosit valorile observate din rularile checkerului:
Testele au prezentat variatii intre:
- Minim: 66/75
- Maxim: 75/75
Speedup observat:
- Pentru 2 thread-uri: intre 1.50x si 2.20x
- Pentru 4 thread-uri: intre 1.90x si 2.84x
Eficienta:
$E(2) = S(2)/2 \rightarrow$ intre 0.75 si 1.10
$E(4) = S(4)/4 \rightarrow$ intre 0.47 si 0.71
Analiza:
Speedup-ul nu este liniar datorita:
- sincronizarii pe structuri comune
- sectiuni de cod care sunt inevitabil secventiale
- overhead I/O la citirea fisierelor JSON
- concurenta pe CPU in cazul testelor in mediu virtualizat
Concluzii:
Scalabilitatea este buna, dar nu perfecta. Programul scaleaza rezonabil pana la 4 thread-uri. Peste 4, castigul este foarte mic datorita naturii I/O-bound a citirii si a sincronizarii necesare pe hashmap-uri.
Cel mai bun numar de thread-uri pentru sistemul meu este 4.