Map and Reduce in a Distributed System

Tomás Costa - 89016 João Marques - 89234

Concurrent Programming with AsyncIO

Usamos a livraria asyncio, uma livraria de programação concorrente pois, já queríamos aprofundar os conceitos abordados na aula e já tínhamos planos futuros de eficiência, algo que o asyncio ajudou com a sua leitura concorrente. Usamos as streams em vez de callbacks, pelo que as mensagens iam sendo lidas e escritas na stream assim que podiam. O asyncio também nos facilitou bastante a distribuição de tarefas, pelo que a programação para múltiplos workers foi trivial. Em termos de performance pensamos que o asyncio apresenta uma vantagem bastante superior a selectors pois retira-nos o problema de criar uma thread por worker.

É importante anotar que desenvolver o projeto com esta livraria foi difícil pois os conceitos de programação assíncrona não são muito fáceis de entender e isso levou a alguma confusão no início, pelo que estávamos a escrever código sincrono no meio do código assíncrono, mas após uma melhor investigação da livraria conseguimos melhorar o código e ajudou-nos bastante na realização do projeto.

Worker (Mapper and Reducer)

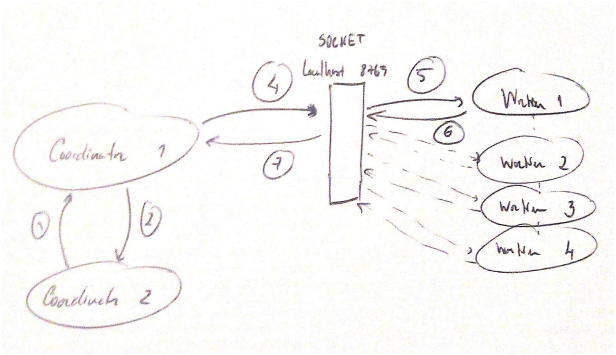
Mapper: O mapper tem apenas uma função map que faz a substituição e tradução de simbolos usando o maketrans que é 2.5x mais rápido que o replace segundo os nosso testes. O array devolve então já ordenado um array com tuplos das palavras do texto e uma ocorrencia á frente, pelo que as palavras iguais ficam seguidas.

Reducer: O reducer tem um papel importante pois é dele que vem uma boa parte da eficiência do código, o objetivo era receber o array devolvido pelo mapper e:

1. Remover repetidos, percorrer a lista e concatenar os repetidos e adicionar a contagem.
2. Receber várias listas mas processar sempre 2 de cada vez.
3. Percorrer a mais pequena com um for, fazer binary search na maior, se encontrassemos um elemento igual devolvemos a sua contagem
4. Fazemos Merge Sort e voltamos a executar o loop se houver mais arrays para iterar, até no fim só sobrar um array reduzido e então o processamento está feito.

Backup Coordinator

- O backup do coordinator foi implementado com base num sistema de troca constante de mensagens do tipo ping/keepalive.
- Ao ser iniciado, se o coordinator deteta que existe outro coordinator a ocupar a socket pré-definida, assume os estado de backup e entra num loop no qual envia periodicamente (cada segundo) uma mensagem ao coordinator principal (1). O coordinator principal responde com uma mensagem que contém o seu estado e todos os dados (2), que são assumidos pelo backup. Caso o coordinator backup detete que o principal morreu (deixou de comunicar), abre novamente a conexão na socket e assume-se como o principal. As últimas mensagens enviadas pelo coordenador principal são consideradas como perdidas, e são reenviadas pelo novo coordenador, assegurando que nenhuns dados são perdidos.



* Coordinator 1 é lançado primeiro, consegue ligar-se ao socket, e assume o estado de principal
* Coordinator 2 não se consegue ligar ao socket e torna-se backup

1: ‘attempt_main’
2: 'coordinator_reply'
4: mensagem guardada numa cache no coordinator caso seja perdida (assumimos que sim no caso da morte do coordinator)

Backup Worker

- Para o backup do worker optámos por guardar, numa cache (no coordinator), a última mensagem enviada para cada worker.
- Quando detetamos que um worker morreu, assumimos que os dados enviados para esse worker foram perdidos. Fechamos a socket de comunicação com esse cliente e enviamos a mensagem perdida para outro cliente.

System Performance

Para além de tentarmos sempre escrever código eficiente e elegante, em muitos casos não relacionados com algoritmos fizemos alguns testes como por exemplo os testes para encontrar um equilíbrio entre os coordenadores perderem mensagens e serem rápidos. Corremos ainda alguns testes com os scripts fornecidos e modificamo-los um pouco para maior visibilidade, tal como o ficheiro run.sh que nos permite correr 4 workers e no fim verificar as diferenças de texto do output.csv para o csv do texto fornecido. Foram realizados 5 testes para 4 workers, e são estas as medias a correr o script chaos_monkey:

Maias: **6.257260 s** Bible: **14.96095 s** Lusiadas: **1.629202 s** Raposa e as Uvas: **0.019799 s**

Estamos bastante satisfeitos com o resultado, para textos maiores como a bíblia perdemos mais algum tempo, mas isso deve-se ao chaos_monkey ter mais tempo para matar processos e as mensagens tem que ser re-enviadas, de todos os testes que efetuamos corremos o comando *diff output.csv $1.csv | grep -e --- | wc -l* para verificar o número de linhas em que erramos e o único ficheiro que obtivemos diferenças foi no texto da biblia em que em algumas execuções perdemos algumas palavras que corresponde a menos de 1%. A adição do backup tornou o código mais lento pois estamos a passar mais mensagens, estas bastante grandes pois são a passagem inteira do estado do coordenador para o backup. No entanto os tempos são na mesma bastante rápidos.