Tutorial do JCL_Big_Sorting

JCL_Big_Sorting é uma aplicação de ordenação distribuída que utiliza a ideia de contagem das frequências para evitar pacotes de dados muito grandes trafegando na rede. Se você não entendeu não se preocupe, vamos explicar com mais detalhes.
Quando temos que ordenar uma coleção de dados, na maioria das vezes o domínio de valores a ser ordenado é pequeno, ou seja, a maioria dos valores se repetem várias vezes. Devido a isso, o JCL_Big_Sorting primeiro conta quantas vezes cada valor aparece para depois ordenar apenas um representante de cada número, tudo isso de forma distribuída.

Vamos analisar o código. O projeto Eclipse precisa apenas ser importado na sua IDE. Basta executar a classe Main.

No nosso arquivo Main.java, dentro do construtor, as primeiras 22 linhas se referem a geração dos arquivos de entrada, que são gerados aleatoriamente. É criado um arquivo de entrada para cada núcleo do cluster.

Assim que os arquivos já foram gerados, é registrado nossa classe Sorting.java, que é responsável pela ordenação, que é dividida em 4 fases. Os argumentos para a primeira fase são criados e a fase 1 é executada em cada núcleo.

A fase 1 é a responsável por ler o arquivo de entrada e fazer a contagem da frequência. Cada número é inserido em uma map chamada values, que contém o número como chave e sua frequência como valor. As chaves da map values são inseridas em uma árvore AVL para ordenação das chaves. Em seguida é criado uma lista de strings que será o retorno da fase 1. Nessa lista são inseridos os valores que são elegidos como pivôs e a frequência acumulada até um pivô em questão. Para ficar claro, todos os números são percorridos um a um fazendo uma soma das suas frequências, quando a soma das frequências de alguns números é maior do que a soma de todas as frequências dividido pela quantidade de núcleos do cluster, o último número lido é elegido como pivô e inserido na lista de strings juntamente com a frequência acumulada até ele. A quantidade de pivôs será mais ou menos equilibrada com a quantidade de núcleos do cluster. Uma variável global é instanciada no cluster com o id do núcleo em questão e a map values. Com isso a fase 1 é finalizada.

Voltando ao Main, a função builInputDataPartitionSchema é chama recebendo como parâmetros todas as listas de strings geradas na fase 1 e a quantidade de núcleos do cluster.
Na função builInputDataPartitionSchema todos os pivôs elegidos na fase 1 são inseridos, com suas frequências acumuladas, em uma map. Note que agora estamos juntando os pivôs de todos os núcleos em uma só map. Os pivôs elegidos por cada núcleo não são necessariamente os mesmos, por isso quando um mesmo número é elegido como pivô em mais de um núcleo suas frequências acumuladas são somadas. Em seguida novos pivôs são elegidos entre o grupo de pivôs da primeira fase. Os novos pivôs são inserido em uma string que será passada como parâmetro na fase 2.

Na fase 2 uma map chamada sorted é criada e recebe a map values da fase 1 que estava salva em uma variável global. Deixando claro que cada núcleo possui sua própria map values. É criado um vetor de maps chamada finais, o tamanho do vetor é a quantidade de núcleos do cluster. Em seguida cada número da map sorted é lido e inserido, com sua frequência, em uma das maps de finais de acordo com os pivôs. Como existe mais ou menos um pivô para cada núcleo, cada map de finais ficará com os números que se encaixam no intervalo de um pivô a outro pivô. Em seguida é criado uma JCL_map, que contém maps, para cada map do vetor finais. A fase 2 é finalizada.

Voltando ao Main, os argumentos para a terceira fase são criados e a fase 3 é executada em cada núcleo. Na fase 3, vamos juntar as maps de cada JCL_map. Como foi criado uma JCL_map para

cada map do vetor finais, e o tamanho do vetor finais é a quantidade de núcleos do cluster, cada núcleo ficará responsável por juntar as maps de uma JCL_map, e cada JCL_map é responsável por um intervalo de pivôs, ou seja, cada uma das maps de uma JCL_map contém os números, com suas frequências, de todos os núcleos nesse intervalo. Os números lidos das maps da JCL_map são inseridos em uma nova map chamada result. Caso exista números iguais em maps diferentes suas frequências são somadas. A fase 3 é finalizada.

Voltando ao Main, os argumentos para a quarta fase são criados e a fase 4 é executada em cada host. A fase 4 apenas verifica se todos os números dos arquivos de entrada existem nas JCL_maps, caso algum número dos arquivos de entrada não esteja nas maps, um erro é contado. Uma função chamada removeDirs é chamada. Essa função exclui todos os arquivos de entrada criados. Voltando ao Main, as variáveis são finalizadas e o JCL_Big_Sorting é finalizado.