다차원 배열 데이터에 대한 편향 인지 분할 기법

김명진, 오소현, 신윤재, 최연정, 이기용
숙명여자대학교 컴퓨터과학부
e-mail : {kimmj, osoh1026, yoonjaeshin, cyj, kiyonglee}@sookmyung.ac.kr

Skew-Aware Partitioning of Multi-Dimensional Array Data

MyeongJin Kim, SoHyeon Oh, YoonJae Shin, YeonJeong Choe, Ki Yong Lee
Division of Computer Science, Sookmyung Women's University

요 약

본 논문에서는 여러 과학분야에서 사용되는 대용량 배열 데이터를 병렬처리를 위해 효율적으로 분할하는 기법을 제안한다. 실제 배열 데이터는 희소(sparse) 배열로 구성된 경우가 많아 기존의 chunking 기법을 사용하면 일부 chunk 에게만 데이터가 밀집되는 편향 현상이 발생하게 된다. 이러한 문제를 극복하기 위해 본 논문에서는 k-d tree 와 유사한 방법으로 공간을 분할하고, 분할된 공간을 chunk 로 두는 방법을 제안한다. 제안 방법에 의해 각 chunk 는 데이터의 밀집 정도가 비슷하게 되어 효과적인 부하분산(load balancing)이 이루어질 수 있다.

1. 서론

최근 들어 물리, 생명, 우주, 지구 등 여러 과학 분야에서 대용량의 배열 데이터가 생성되고 있다[1]. 이러한 대용량 배열 데이터는 효율적인 분석을 위해 다수의 컴퓨터에 분할되어 병렬로 처리되어야 한다. 이를 위해 배열 데이터를 분할하는 Regular chunking, Directional tiling, Sliced chunking, Worklaod-based chunking 등 다양한 기법이 제안되었다[2][3][4].

실제 배열 데이터는 희소(sparse) 배열로 구성된 경우가 많다. 여기서 말하는 희소 배열은 일부 원소만 값을 가지고 있고 대부분 원소의 값은 정의되어 있지 않은 배열을 말한다. 이 경우에 기존의 chunking 기법을 사용하면 일부 chunk 에게만 데이터가 밀집하게 되는 편향 현상이 발생하게 된다[5]. 따라서 본 논문에서는 각 chunk 에 포함된 데이터의 밀집 정도가 비슷하도록 chunk 를 분할하는 기법을 제안하고자 한다.

제안 방법은 k-d tree 와 유사한 방법으로 공간을 분할하고[6], 분할된 각 공간을 chunk 로 두는 것이다. 제안 방법을 통해 각 chunk 는 데이터의 밀집 정도가 비슷하게 되어 효과적인 부하분산(load balancing)이 이루어질 수 있다.

본 논문의 구성은 다음과 같다. 2 장에서는 문제를 정의하고, 3 장에서는 기존의 chunking 기법에 관한 관련 연구를 살펴본다. 4 장에서는 논문에서 제안하는 분할 방법을 자세히 설명하고, 5 장에서는 제안 방법을 분석한다. 마지막으로 6 장에서 결론 및 추후 연구를 기술한다.

2. 문제 정의

본 논문에서는 2 차원 이상의 다차원 배열을 고려한다. A_n 을 n 차원 배열이라 하자. 배열 내부의 각 원소는 null 값을 가지거나 특정한 값을 가진다. 이 때 A_n 의 각 원소를 $a_{ijk...}$로 표현하며 i, j, k, …는 각 차원의 첨자를 의미한다.

A_n 을 분할한 각 chunk 는 그의 upper-left 의 좌표와 lower-right 의 좌표를 사용하여 ($[l_1, h_1, \dots]$, $[l_2, h_2, \dots]$, …. , $[l_n, h_n, \dots]$)와 같은 형태로 나타낸다. 예를 들어 2 차원 배열 A_2 를 분할한 각 chunk 는 ($[l_1, h_1]$, $[l_2, h_2]$)과 같은 형태로 표현된다. 또한 각 chunk 는 실제 값이 있는 원소들만 (좌표, value)의 형태로 나열되어 저장된다고 가정한다. 즉, 2 차원 배열의 원소 a_{ij} 는 그의 값이 v 일 경우 (i, j, v) 형태로 저장된다.

본 논문에서 제안하는 방법은 A_n 의 모든 원소들 중 값이 정의된 원소의 개수를 N 이라 할 때, A_n 을 여러 chunk 로 나누되 N 개의 원소가 이들 chunk 에 고르게 분포되도록 분할한다. 본 논문에서는 각 chunk 가 최대 k 개까지의 원소를 저장할 수 있고 k 는 N 보다 작은 수라 가정한다. 제안 방법은 각 chunk 가 포함하고 있는 원소가 k 개 이하가 될 때까지 분할을 진행한다.

3. 관련 연구

배열 데이터를 분할하는 방법에 대해서는 다양한 기법이 제안되어왔다.

배열을 chunk 로 나누는 가장 간단한 방법은 Regular chunking 이다. Regular chunking 은 각 차원을 동일한 크기의 세그먼트(segment)로 나누고, 배열을 각 차원에 대해서 세그먼트 단위로 분할한다. 그러면 배열은 모든 차원에 대해 동일한 크기를 가지는 hyper-cube 들로 분할되며, 각 hyper-cube 가 하나의 chunk 가 된다. 각 차원에 대한 세그먼트 수는 차원에 관계없이 동일한 수가 되도록 할 수도 있고, 차원

에 따라서 서로 다른 수가 되도록 할 수 있다.

Directional tiling 은 각 차원에 대해 분할할 위치를 서로 독립적으로 결정하고, 분할 위치에 따라 배열을 분할하는 방법이다[2]. 따라서 chunk 는 서로 다른 크기와 모양을 가질 수 있다. 하지만 이 경우 가장 큰 chunk 의 크기가 디스크 접근 단위를 넘을 경우 추가로 분할을 진행해야 하며, 어떤 chunk 의 크기가 디스크 접근 단위보다 작은 경우 여러 chunk 을 묶어 하나의 chunk 로 만들 수도 있다.

Sliced chunking 은 배열을 특정한 한 차원을 기준으로 분할하는 방법이다. 따라서 분할결과는 분할 기준이 된 차원의 도메인 크기만큼의 개수를 갖는 N - 1 차원의 hyper-cube 들이 된다. 각 hyper-cube 들은 서로 독립적으로 더 분할될 수 있다.

Workload-based chunking 은 배열의 원소를 접근하는 질의 패턴에 따라 배열을 분할하는 방법이다[3][4]. 최악의 경우에는 배열의 특정 원소들을 얻어내기 위해 모든 chunk 를 읽어야 할 수도 있고, 최선의 경우에는 단 1 개의 chunk 을 읽어 원하는 모든 원소들을 얻어낼 수도 있다. 따라서 이 방법은 빈번하게 발생하는 질의 패턴을 파악하고, 그들이 접근하는 원소들을 얻어내기 위해 읽어야 할 chunk 의 수가 최소화되도록 배열을 chunk 로 분할하는 방법이다.

하지만 이 방법들은 희소 배열에 대해 데이터 편향을 고려한 분할 방법을 고려하고 있지 않다. 따라서 다음 절에서는 본 논문에서 제안하는 데이터 편향을 고려한 분할 방법을 자세히 설명한다.

4. 제안 분할 방법

이 논문에서 제안하는 파티셔닝(partitioning) 방법은 크게 두 단계로 구성된다. 첫 번째 단계에서는 k-d tree construction 방식과 유사한 방식으로 공간을 분할한다[6]. 주어진 n 차원의 배열 데이터를 입력으로 하며, 각 차원을 순차적으로 돌아가며 기준으로 삼아 현재 파티션에 포함된 배열데이터의 사이즈를 이분(二分)하는 값의 정보를 2 진 트리 형태로 저장한다. 이를 간단한 pseudocode 로 나타내면 Algorithm 1 과 같다.

Input: n-dimensional array data A_n
Output: a binary tree with nodes including partitioning information.

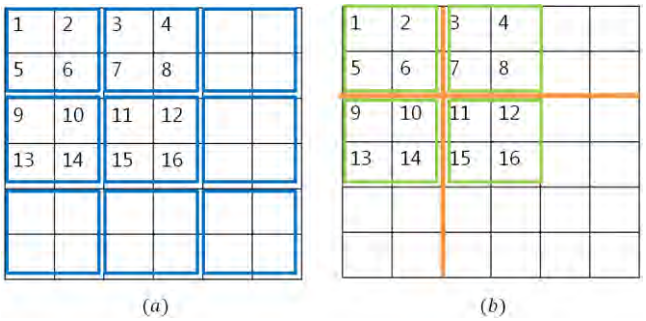
```
function partTree(An, int depth){
   while(sizeof(An)>k){

   var int axis :=depth mod n;

   select dividing_value by axis from An

   var tree_node node;
   node.location := dividing_value;
   node.leftchild := partTree(points in An before dividing_value,
   depth+1);
   node.rightchild := partTree(points in An after dividing_value,
   depth+1);
   return node;
   }
}
```

Algorithm 1: Skew-Aware Partitioning Algorithm

다음 단계에서는 각각의 배열 데이터를 앞 단계에서 만들어진 트리를 적용하여 어떤 파티션에 속하는지 판단하여 해당 파티션에 추가한다.

5. 제안 방법 분석

그림 1 은 (a) 기존의 방법인 Regular chunking 과 (b) 제안 방법의 간단한 예다. 데이터 편향현상이 일어나는 상황에서 기존의 방법은 무조건 동일한 규격으로 분할하고, 본 논문에서 제안된 분할 방법은 데이터가 밀집된 부분을 집중적으로 분할하는 모습이다.

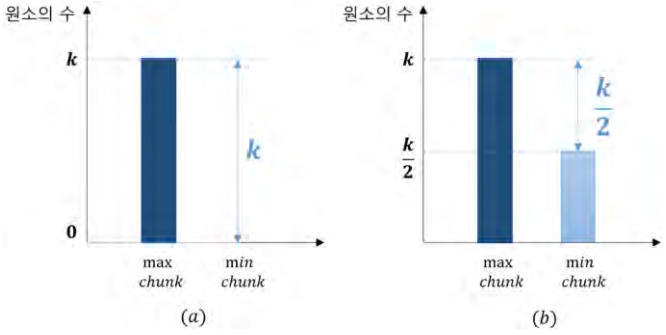


(그림 1) (a) Regular chunking 과 (b) 제안 방법의 간단한 예

기존의 방법과 제안 방법을 비교하기 위해 최악의 경우를 분석한다. 기존의 방법으로 N 개의 원소를 regular 한 M 개의 chunk 로 나누는 방법을 생각해 볼 수 있다. 최대로 원소가 많이 포함된 chunk 의 원소 수는 k 개 이고 어떤 chunk 에는 원소가 하나도 존재하지 않는다. 따라서 각각의 chunk 가 노드로 전달 되면 N 이 매우 클 경우, 노드 간 처리량의 차이가 발생하게 되어 질의 실행 시간(query execution time)에 영향을 미치게 된다.

하지만 제안 방법을 사용한다면 위의 경우를 피할 수 있다. N 이 짝수의 곱으로만 이루어진 경우는 데이터의 개수를 고르게 나누므로 고려할 필요가 없다. N 에 한번이라도 홀수가 곱해진다면 분할을 한 번 진행했을 때 데이터의 수는 ([N/2]+1)개와 ([N/2])개로 나뉘게 된다. 이 때 만약 ([N/2])이 k 라면 ([N/2]+1)은 k 를 초과하므로 한번 더 분할이 진행 된다. 따라서 원소가 가장 많이 들어간 chunk 와 가장 적게 들어간 chunk 의 원소 수의 차이는 약 k/2 라 할 수 있다. 분할이 더 진행 되었더라도 원소가 가장 많이 들어간 chunk 와 가장 적게 들어간 chunk 의 원소 수의 차이는 약 k/2 이다. 기존의 방법보다 하나의 chunk 에 들어가는 데이터의 수가 고르게 분포되므로 노드 간 데이터 처리량의 차이가 줄어들어 병렬성을 높일 수 있다.

아래 그림 2 에 위의 내용을 간략히 나타내었다. X 축의 max *chunk* 는 원소가 가장 많이 들어가는 chunk 를 의미하며 min *chunk* 는 원소가 가장 적게 들어가는 chunk 를 의미한다. Y 축은 한 chunk 에 들어가는 원소의 수를 의미한다. (a)는 Regular chunking 의 worst-case 를 나타냈으며 (b)는 제안방법의 worst-case 를 나타내었다.



(그림 2) (a) Regular chunking 과 (b) 제안방법 비교

6. 결론 및 추후 연구

본 논문은 대용량 배열 데이터에서 부하분산(load balancing)을 고려한 chunking 방법을 제안하였다. 제안 방법은 k-d tree 와 유사한 방식으로 공간을 분할하고, 분할된 각 공간을 chunk 로 두어 각 chunk 의 데이터 밀집 정도가 비슷하게 분포되도록 한다.

추후 연구로는 본 논문의 이론적 분석을 바탕으로 실제 질의 처리 성능을 측정하여 제안 분할 방법의 부하 분산 효과를 분석할 것이다.

Acknowledgement

이 논문은 2015 년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. 2015R1C1A1A02037071)

참고문헌

[1] Jim Gray, David T. Liu, Maria Nieto-Santisteban, Alex Szalay, David J. DeWitt, Ger Heber, “Scientific Data Management in the Coming Decade", ACM SIGMOD Record, vol.34, no. 4, pp. 34-41, 2005.

[2] P. Furtado and P. Baumann, "Storage of Multidimensional Arrays Based on Arbitrary Tiling". In Proceedings of 1999 IEEE ICDE International Conference on Data Engineering, pp. 480-489, 1999.

[3] S. Sarawagi and M. Stonebraker, "Efficient Organization of Large Multidimensional Arrays," In Proceedings of 1994 IEEE ICDE International Conference on Data Engineering, pp. 328-336, 1994.

[4] E. J. Otoo, D. Rotem, and S. Seshadri. "Optimal Chunking of Large Multidimensional Arrays for Data Warehousing," In Proceedings of 2007 ACM DOLAP International Workshop on Data Warehousing and OLAP, pp. 25-32, 2007.

[5] Jennie Duggan , Olga Papaemmanouil , Leilani Battle , Michael Stonebraker, ” Skew-Aware Join Optimization for Array Databases”, ACM SIGMOD, pp. 123-124, 2015.

[6] https://en.wikipedia.org/wiki/K-d_tree