병렬 연관규칙 마이닝을 위한 동적 부하 분산 설계 및 구현

김현민° 김지혜 안창욱 R.S. Ramakrishna
광주과학기술원 정보통신공학과
{hmkim, jhkim, cwan, rsr}@kjist.ac.kr

Effective Dynamic Load Balancing for Association Rule Mining

Hyun-Min Kim° Jihye Kim Chang Uk An Ramakrishna
Dept. of Information & Communication
Kwang-Ju Institute of Science and Technology(K-JIST)

요 약

데이터 마이닝 기술 중 하나인 연관규칙 마이닝의 병렬 알고리즘들은 동형질의 병렬 컴퓨팅 시스템을 대상으로 하여 개발되었다. 그러나, 이러한 병렬 알고리즘들은 클러스터 시스템 또는 Network Of Workstation(NOW)과 같은 저가의 프로세서들로 구성된 집합적인 병렬 컴퓨팅 시스템에서는 부적당하다. 이는 이들 시스템이 다른 성능을 가진 프로세서로 구성되어 있거나 여러 사용자의 접근을 허용하는 등의 이형성을 가지기 때문이다. 결과적으로 이러한 환경을 고려하지 않은 기존의 병렬 연관규칙 알고리즘들은 전체 시스템의 성능을 활용하지 못하게 되어 성능저하를 피할 수 없다.

본 논문에서는 대표적인 병렬 연관규칙 알고리즘인 Data Distribution 알고리즘을 위한 효과적이고 확장성 있는 동적 부하분산 알고리즘의 설계와 구현을 다룬다.

1. 서 론

데이터 마이닝은 대용량의 데이터 베이스 또는 가공되지 않은 데이터로부터 유용하고 가치 있는 정보 또는 지식을 추출하는 제반 과정을 말한다. 데이터 마이닝 기법들은 공통적으로 데이터의 양적 증가에 따른 확장성 문제를 해결하기 위해 병렬화가 필수적이다. 특히, 연관규칙 마이닝을 위해 다양한 병렬화 기법들이 연구 되었다. 대부분의 병렬 연관규칙 마이닝 알고리즘은 동일한 성능의 노드를 고려하여 개발되었다. 이러한 동형질의 병렬 컴퓨팅 환경은 주로 고가의 슈퍼 컴퓨터를 기반으로 한 것이다. 최근에는 고성능 네트워크의 개발과 PC나 Workstation의 안정성과 성능의 향상으로 저렴한 가격으로 슈퍼 컴퓨터 이상의 성능을 보이는 클러스터링 기법들이 선보이고 있다. 그러나, 클러스터 시스템의 특성상 성능의 차이를 보이는 프로세서나 여러 사용자의 접속 허용에 따른 이형 컴퓨팅 환경을 피할 수 없다. 결과적으로 기존의 병렬 연관규칙 알고리즘은 이형 시스템환경에서 확장성이 떨어지는 현상을 보인다.

본 논문에서는 PC-Cluster및 NOW등의 집합적 컴퓨팅 환경에서 병렬 연관규칙 알고리즘의 확장성 보장을 위한 동적 부하분산 알고리즘의 개발을 다룬다. 부하의 효과적인 예측을 위한 Dynamic Support Count 기법과 함께 대표적인 병렬 연관규칙 알고리즘인 DD(Data Distribution algorithm)[1]를 위한 맞춤식 부하분산 기법을 제안한다. 이형 컴퓨팅 환경을 위해 PC-Cluster와 NOW를 혼합한 시스템이 구축되고 다사용자 환경이 모의로 구현되어 성능 평가에 사용된다.

2. 관련 연구

2.1 연관규칙 알고리즘

연관규칙 마이닝의 대표적인 알고리즘인 apriori[1]를 백화점 데이터를 예를 들어 설명하겠다. 수 십만 또는 수 백만의 고객 상품 정보가 데이터 베이스에 저장되었을 때, primary key를 고객의 id로 하여 고객이 구입한 item에 대한 itemset을 tuple로 하는 레코드를 생각해 보자. 연관 규칙이라 함은 item 또는 itemset에 대해서 다른 item또는 itemset이 함께 구입 될 확률이 기준치 이상인 규칙을 의미한다. 일례로 도자기와 TV를 구입한 고객이 맥주를 구입할 확률이 기준치 이상인 경우 이것을 {도자기, TV} -> {맥주} 의 형태의 규칙으로 표현한다. [그림 1]은 정형화된 연관규칙 마이닝 알고리즘이다. Frequent k-itemset(L_k)은 레코드의 transaction에 빈출하는 item의 개수가 k인 item set을 의미하며 Candidate itemset(C_k)은 빈출할 가능성이 있는 itemset을 의미한다. Apriori algorithm은 크게 빈출 itemset을 찾는 단계와 규칙생성 단계가 있으며 대부분의 컴퓨팅 시간은 첫 번째 단계에서 소요 된다.

2.2 연관규칙 알고리즘의 병렬화

많은 병렬 연관규칙 알고리즘들이 기본적으로 다음 중 하나의 병렬화 정책을 따른다. 연관규칙 알고리즘에서 의미하는 두 가지 병렬화 정책인 task parallelism과 data parallelism은 일반적 의미와 차이가 있다[2]. 연관규칙은 전형적으로 dataset으로부터 candidate concept을 구성하고 그 concept을 다시 dataset으로 support 하는 형태를 취한다. Task parallelism을 사용하는 알고리즘은 concept을 각각의 노드에 나누어 동일 dataset으로 support한 후 disjoint한 concept들을 합치는 방법을 따르고, data

```
Step 1   Frequent Pattern finding :
  L1 = { frequent 1-itemsets };
  for (k =2; Lk-1 ≠Ø; k++ ) do begin
        Ck= apriori-gen ( Lk-1 );   // New candidates
        forall transactions t ∈ D do begin
             Ct   = subset ( Ck, t);//Candidates contained int
             forall condidates c ∈ Ct   do
                        c.count ++;
        end
        Lk   =   { c ∈ Ck | c.count ≥ minsup }
  end
  Patterns = ∪k Lk ;
Step 2   Rule Generation:
  Rules = rule-generate( Patterns )
```

그림 1. apriori algorithm

parallelism의 경우에는 dataset을 나누어 동일한 concept에 대해 부분적인 support를 한 후, 다른 노드와의 support를 합쳐서 global support를 얻는 방법을 취한다. Task parallelism의 대표적인 알고리즘으로는 distributed memory를 기반으로 하는 DD,IDD,[3][4] 등이 있으며, data parallelism을 취하는 것에는 distribute memory를 기반으로 하는 CD,FPM,[3][5] 와 shared memory를 고려한 CCPD[6] 등이 있다.

3. DD를 위한 동적 부하분산

앞에서 논의한 바와 같이 연관규칙의 concept과 dataset은 연관규칙 알고리즘의 연산 대상이자 작업부하로서 작용한다. 기존의 병렬 연관규칙 알고리즘은 정적 부하분산 방법을 통해 성능 향상을 꾀했다고 볼 수 있다. DD와 같은 Task parallelism식의 알고리즘은 concept을 노드의 성능에 맞게 분배하는 것에 초점을 맞춘 반면 CD와 같은 data parallelism에서는 dataset을 프로세서의 성능에 맞춰 분배한 것이다. 이 같은 부하분산 방법들은 부하를 여러 프로세서에 나누기 전에 분배할 부하량을 결정해야 하는 어려움이 있다. 부하를 정적인 방법으로 균등하게 분배하기 어려운 이유는 근본적으로 dataset의 transaction(tuple)의 패턴이 예측 불가능 하기 때문이다. 서로 다른 패턴을 가진 transaction이 concept을 support 할 때—concept의 자료구조 hash tree, prefix tree, FP tree등은 transaction이 candidate itemset을 빠르게 접근하도록 도와준다—각각의 candidate itemset에 접근 하는 횟수가 부하로 작용한다. 그러나, 남은 transaction들의 패턴을 알 수 없기 때문에 얼마나 많은 transaction들이 접근할 것인가를 정확히 예상할 수가 없다.

3.1 동적 부하 분산 설계

부하분산의 네 단계는 부하 측정• 상태정보 교환• 분산 게시• 부하 전송 이다. 각 단계별로 DD에 적합한 *periodic global distributed symmetric* 방법[8]을 선택하였다. DD의 특성을 이용해 data exchange packet[1]에 덧붙이는 piggy-back 방법을 취해서 상태정보 교환을 위한 추가 통신비용을 줄였다. 이 정보는 각각의 프로세서에 분산(*distributed*)되어 있는 load balancer에게 *periodic*하게 전달되어 분산 게시 및 분산 결정의 근거로 사용된다. 분산 결정은 부하분산 결정 정책(*symmetric policy*, 3.3참조)에 따라 수행된다.

표 1. 기호 표기

P	*number of processors*
C	*concept amount*
C_i	*local concept amount*
N	*number of total transactions*
N_i	*number of processor i's local transactions*
N_i^l	*number of processor i's remained local transactions*
N_i^e	*number of processor i's remained external transactions*
T_i^r	*processor i's remained time*
T_i^t	*processor i's mean time for one transaction to traverse concept*
T^o	*workload transfer overhead time*
T^h	*hash tree construction time*
$S_i=C_i/T_i^t$	*processor i's effective speed*
τ	*average round-trip communication time for a transaction exchange*
γ	*average round-trip communication time for a workload exchange*

3.2 동적 부하 예측

앞에서 제기한 문제를 풀기위해 다음과 같은 효과적인 부 하 예측 방법을 제안한다. 우선 concept amount(C)를 다음과 같이 정의 한다. 지금부터의 기호표기에 대한 설명은 [표 1]을 참조한다.

$$C = \sum_{i \in M} s(i) - d(i) \qquad (1)$$

for candidate itemset i,
M: set of candidate itemsets; s(i)-d(i): dynamic support count;
s(i): minimum support count of subset of i, static support count;
d(i) currently supported count of i.

위에서 제안한 Dynamic Support Count(DSC)는 기존의 static support count[7]와 달리 support의 진행에 따라 감소하는 잠재적 support횟수(concept의 weight)를 고려한다.

부하지수(load index)는 CPU , memory, I/O speed를 함축적으로 반영하는 transaction당 평균 응답시간(T_i^t)을 사용했다. 다시 말해 프로세서가 하나의 transaction에 대해 자신의 concept(C_i)을 support하기 위해 소요한 평균시간이다. 이를 기본으로 각각의 프로세서 i에 대해 effective speed(S_i)를 다음과 같이 정의한다.

$$S_i = C_i / T_i^t \qquad (2)$$

3.2 부하 분배 정책

Idle time을 배제한 프로세서 i에서의 남은 실행 시간(T_i^r)을 다음과 같이 예상할 수 있다.

$$T_i^r = (N_i^l + N_i^e) \times T_i^t + N_i^e \times \tau \qquad (3)$$

또한, DD에서는 concept(C_i)이 이동 가능한 부하 단위 이므로 concept변화량 ΔC_i 에 따른 남은 시간 ΔT_i^r 에 대한 수식을 얻을 수 있다.

$$\Delta T_i^r = \frac{(N_i^l + N_i^e)}{S_i} \times \Delta C_i \qquad (4)$$

$$\Delta T_i^r = \bar{T}_i^r - T_i^r = (N_i^l + N_i^e) \times (\bar{T}_i^t - T_i^t)$$

$$T_i^t = C_i / S_i, \bar{T}_i^t = \bar{C}_i / S_i, \Delta C_i = \bar{C}_i - C_i$$

식(3)을 이용하여 프로세서들의 남은 실행시간의 평균(T_{ave}^r)에 대한 분배 불균형으로 인한 다른 프로세서들의 지연 시간에 대한 차이를 기준으로 부하 분배에 대 한 결정을 한다. *improvement _ ratio* 는 10%를 실험적으로 사용하였다.

$$improvement\_ratio = 1 - \frac{T_{ave}^r}{MAX(T_i^r) + T^o} \qquad (5)$$

$T^o = \gamma \times |\Delta C| + \alpha T^h$, DLB overhead
Δ C : the number of total candidate itemsets to be transferred
α : the ratio of number of candidate itemsets to be transferred over the |Δ C|

4. 실험결과

CD, DD 및 Dynamic Load Balancing algorithm(DLB)은 C++/MPI로 구현하였고 실험 데이터는 T10I2D1000[17]를 사용하였다. 시스템 환경은 3대의 NOW(spacs 64MB) 와 3대의 PC-Cluster(Pentium III, 733MHZ, 128MB)를 사용하였다. 네트워크는 100 Mbps fast Ethernet (1615.6 μs latency, 88.06 Mbps bandwidth)이다.

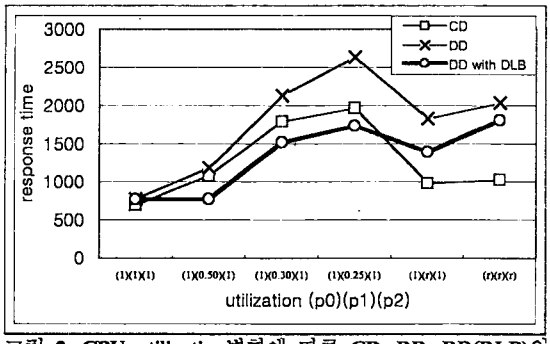

그림 2. CPU utilization변화에 따른 CD, DD, DD(DLB)의 수행 시간 비교 (x) = x% utilization, (r) = changeable random number

첫 번째 실험 결과는 3-NOW 시스템에서의 다사용자 환경에 대한 성능평가 이다. 다사용자 환경을 모의 실험하기위해 각각의 프로세서의 CPU utilization에 변화를 주었다(i.g. (1)(r)(1)은 2대의 프로세서의 utilization이 100%이고 나머지 한대의 utilization은 25%~100%사이에서 200 sec. 마다 변화함을 의미한다). [그림 2]는 stable balanced((1)(1)(1)), stable imbalanced((1)(x)(1), x는 상수), unstable balanced((r)(r)(r)) 환경에서의 성능 비교이다. Stable imbalanced 환경에서는 imbalance 정도가 높을 수록 부하분산(DLB)을 사용하였을 때의 성능이 더 좋은 것을 볼 수 있다. 동적 부하분산의 부하분산 오버헤드가 가장 큰 unstable balanced 환경에서도 10%이상의 성능 향상을 보인다.

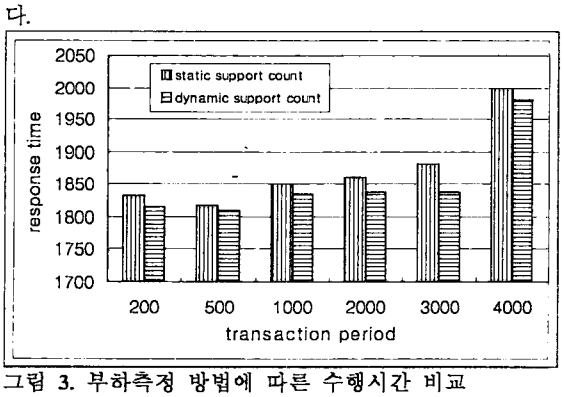

그림 3. 부하측정 방법에 따른 수행시간 비교

[그림 3]은 unstable balanced환경에서 static support count 방법과 DSC 방법을 비교 평가한 것이다. 성능 정보의 monitoring 주기를 변화 시키면서 측정 하였을 때 최적의 주기(매 500 transaction마다)에서 뿐만 아니 라 다른 주기에서도 DSC 방법이 상대적으로 좋은 성능을 보인다. 이

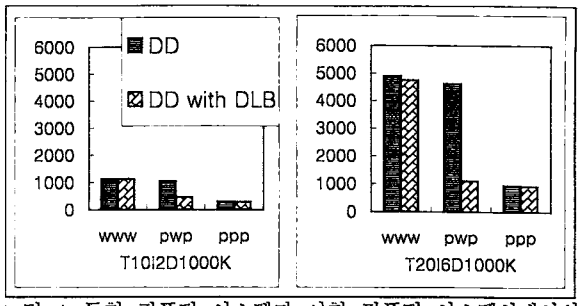

그림 4. 동형 컴퓨팅 시스템과 이형 컴퓨팅 시스템상에서의 동적 부하분산 적용여부에 따른 성능 비교, w: workstation, p : PC, i.g. pwp : 1개의 workstation과 2대의 PC-cluster 시스템

러한 결과는 DLB의 부하예측이 더 효과적이어서 ping-pong effect와 같은 불필요한 부하이동이 덜 발생함을 의미한다.

마지막 실험[그림 4]은 PC-Cluster와 NOW의 혼합형인 이형 시스템 환경에서의 성능 평가이다. DLB를 사용한 DD 프로그램이 그렇지 않은 프로그램에 비해 전체 시스템의 컴퓨팅 파워를 사용하는 능력이 향상됨을 볼 수 있다.

5. 결론

본 논문에서 제안한 동적 부하분산 알고리즘을 병렬 연관규칙 알고리즘에 적용하여 이형환경의 컴퓨팅 시스템의 성능을 활용하고 알고리즘의 확장성을 높일 수 있었다. 부하분산의 효과적인 예측을 위해 Dynamic Support Count(DSC)를 제안함과 아울러 DD 맞춤형 부하분산 알고리즘을 설계하고 구현하였다.

다사용자 환경을 모의 실험 결과로 컴퓨팅 시스템 환경이 stable imbalance한 경우에서 30%이상의 성능 향상과 unstable balance한 경우에도 10%이상의 성능을 보였다. NOW와 PC-Cluster의 혼합형이 이형 환경에서의 실험에서 DD에 제안된 부하분산 방법을 적용하여 낮은 성능을 보이는 NOW에 따른 전체 성능저하의 문제를 해결하였다.

6. 참고 문헌

[1] R. Agrawal, R. Srikant: "Fast Algorithms for Mining Association Rules", Proc. of the 20th Int'l Conference on Very Large Databases, Santiago, Chile, Sept. (1994).

[2] Mohammed J. Zaki, "Parallel and Distributed Association Mining: A Survey," in IEEE Concurrency, special issue on Parallel Mechanisms for Data Mining, Vol. 7, No. 4, pp14-25, December, (1999).

[3] R. Agrawal, J.C. Shafer: "Parallel Mining of Association Rules", IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 6, December (1996).

[4] Han, E.-H.; Karypis, G.; and Kumar, V. "Scalable parallel data mining for association rules," In ACM SIGMOD Conference Management of Data, (1997).

[5] D.W. Cheung and Y. Xiao, Effect of Data Distribution in Parallel Mining of Associations, In Data Mining and Knowledge Discovery, 3, 291-314, 1999.

[6] M.J. Zaki et al., "Parallel Data Mining for Association Rules on Shared-Memory Multi-Processors," Proc. Supercomputing '96, IEEE Computer Soc. Press, Los Alamitos, Calif, (1996).

[7] Masahisa Tamura and Masaru Kitsuregawa, "Dynamic Load Balancing for Parallel Association Rule Mining on Heterogeneous PC Cluster System," Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, (1999).

[8] Load Distribution: a Survey- Santos(1996).