인터넷에서 Web Proxy의 최적 개수와 위치 : 트리 구조

정행은[1], 최정임[2], 이상규[3], 문봉희[4]
haengeun@cs.sookmyung.ac.kr[1], jungim@cs.sookmyung.ac.kr[2],
sanglee@cs.sookmyung.ac.kr[3], moon@cs.sookmyung.ac.kr[4]
숙명여자대학교 전산학과

Optimal Number and Placement of Web Proxies in the Internet : The Tree Topology

Haeng-Eun Chung, Jung-Im Choi, Sang-Kyu Lee and Bong-Hee Moon
Department of Computer Science, Sookmyung Women's University

요 약

최근 기하급수적으로 증가하는 WWW(World Wide Web)의 사용으로 인하여 전체적인 네트워크의 성능이 저하되어 서버가 클라이언트에 빠른 서비스를 할 수 없는 경우가 자주 발생되고 있다. 이러한 서비스 지연의 해결방안으로 web proxy를 사용하여 서버의 부하와 traffic을 줄이는 방법을 많이 사용하고 있다. 이와 같은 web proxy는 인터넷상에서의 위치에 따라 그 효용이 달라지게 되는데, 이와 관련하여 [5] 에서는 트리 구조의 인터넷상에서 전체적인 access time을 줄이기 위해 미리 정해진 개수(M)의 proxy의 위치를 구하는 방법에 대해 설명하고 있다. 하지만 서버 관리자의 측면에서는 서버를 구축하고 유지하는데 드는 비용과 직결되는 필요한 proxy의 개수를 정하는 것도 중요한 문제이다. 그러므로 본 논문에서는 인터넷상에서 트리 구조의 서비스 노드들 각각이 정해진 임계 응답시간을 만족하기 위해 필요한 최소한의 proxy의 개수와 그의 위치를 동시에 찾아내는 알고리즘을 제안한다.

1. 서론

지난 몇 년 간 WWW(World Wide Web)과 관련된 도구들의 개발과 그 도구들의 사용이 용이하게 됨에 따라 WWW의 이용자수는 급격히 증가하였다. 그러나 제한된 리소스(resource)로 인해 전체적으로 네트워크(network)의 성능이 저하되어 클라이언트(client)로부터 전달되는 데이터 요구에 서버(server)가 제때에 응답을 해주지 못하는 경우가 자주 발생되고 있다. 이와 같은 문제는 특히 서버나 네트워크의 혼잡(congestion), 부적절한 대역폭(bandwidth)을 가지는 링크, 그리고 propagation delay등으로 인해 야기되는데 이러한 원인을 제거하기 위해 일반적으로 웹 캐싱(web caching)기법을 이용한다. 캐싱이란 클라이언트가 서버보다 가까운 위치에서 정보(document)를 가지고 오게 하는 방법으로 이를 통해 서버의 부하를 줄이고 propagation delay time이 줄어드는 효과를 가져온다. 캐싱이 이루어지는 곳으로는 web 서버 자신과 web proxy 그리고 클라이언트 상에서의 web browser등이 있다[1]. 클라이언트 위치에서의 캐싱은 클라이언트의 web browser에서 이루어지는데, 이는 짧은 시간 동안 한 클라이언트가 동일한 장소를 가는 경우에는 traffic을 줄일 수 있지만 여러 클라이언트가 같은 위치의 정보를 요구하는 경우에는 서로 도움을 주지 못한다[2]. web 서버의 위치에서 이루어지는 캐싱은 web 서버가 다른 서버로의 포인터를 가지는 방법[1]으로 클라이언트의 요구에 local copy fetching을 이용하여 응답을 보내서 요구를 더 이상 하진 않지만 잠재적으로 긴 propagation delay이나 혼잡을 완화하지 못한다[3]. 전체의 지체(latency)를 줄이는 가장 효율적인 방법은 web proxy를 이용하는 것인데 web proxy란 클라이언트와 서버사이에 캐싱을 위해 둔 중간 서버로서 이는 서버의 부하를 줄이는 효과를 가져온다. 그러므로 web proxy를 통한 캐싱이 가장 많이 쓰이고 있으며 이에 관한 연구가 활발히 이루어지고 있다[4][5]. 이에 본 논문은 트리 구조(tree topology) 네트워크에서 주어진 조건을 만족하도록 하면서 클라이언트의 요구에 응답할 수 있는 범위 내에서 필요한 최소의 web proxy 개수를 구하고, 이들의 알맞은 위치를 찾는 방법을 제시한다.

이와 관련된 연구로 인터넷에서 web proxy의 개수가 정해져 있을 때 최적의 위치를 찾는 것[4]이 있는데, [4]는 이 논문에서 진술했듯이 전체 traffic과 latency 두 가지 요소를 고려하여 web proxy들을 적절한 곳에 위치시킴으로서 네트워크 리소스와 traffic 형태를 고르게 분포시켜서 전체 네트워크가 지체되는 것을 최소화하는데 그 목적이 있다. 그 논문에서는 간단한 linear구조를 고려하여 web proxy의 개수가 M, 노드의 개수가 N으로 주어졌을 때 $O(N^2M)$의 시간 복잡도(time complexity) 안에 다이나믹 프로그래밍 알고리즘 (Dynamic Programming Algorithm)을 이용하여 문제점을 해결했다. 또 이와 관련하여 그들은 같은 문제를 트리 구조에 응용하여 해결해 낸 방법[5]을 제시했다. 이 논문 역시 다이나믹 프로그래밍 문제를 모델링하여 트리 구

조에서 $O(N^2M^3)$의 시간 복잡도 안에 M개의 주어진 web proxy를 N 개의 노드에 가장 적절하게 배치하는 알고리즘을 구하였다. 위의 문제들은 주어진 개수의 web proxy를 네트워크에 배치하기 위한 최적의 위치를 찾는 알고리즘을 기술한 반면, 본 논문에서는 web 서버 관리자가 클라이언트에게 어느 한도 지연시간을 초과하지 않는 QoS를 제공하는데 필요한 최소한의 web proxy의 개수와 그들의 배치를 찾는 방법을 제안한다. 이는 web 서버의 관리가 클라이언트의 요구를 만족하면서 web proxy의 개수를 최적화 하여 시스템 구축과 유지비용을 최소화할 수 있다는데 의의를 가지고 있다.

본 논문의 구성은 다음과 같다. 2장에서는 트리에서의 주어진 조건을 만족하기 위한 최소한의 proxy의 개수와 그것들의 위치를 구하는 방법 및 알고리즘을 살펴보고 3장에서 결론을 기술한다.

2. 문제 해결 방안 및 기법

본 논문에서 제시한 문제 해결 방안은 트리 구조(tree topology)를 기반으로 한 것인데 기본 모델은 <그림 1>과 같다.

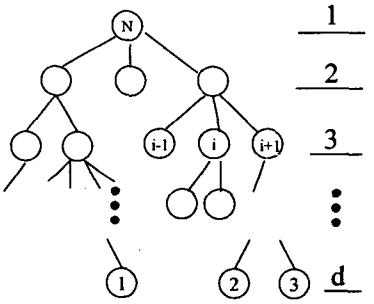

그림 1 트리 구조 모델

N 개의 노드로 이루어진 트리에서 임의의 노드를 i 라 하면 (이 때, $1 \leq i \leq N$), 노드 i는 서버로부터의 거리 차이 때문에 생기는 propagation delay ($P[i]$) 값과 자신과 자신의 하위의 노드들 때문에 생기는 traffic에 대한 percentage ($T[i]$) 값을 가지고 있다. 다시 말해 traffic percentage $T[i]$는 노드 i를 통해 web서버에 접근하는 전체 traffic의 비율을 뜻하므로, $T[i]$값은 노드 i 의 부모 노드의 traffic percentage 값보다 항상 작거나 같다. propagation delay $P[i]$는 web서버에서 노드 i 까지의 거리에 비례하므로 트리 모형에서는 root에서 leaf 노드로 갈수록 그 값은 커지게 된다.

각 노드의 비용은 그 노드가 leaf 노드인지 아닌지에 따라 다르게 계산된다. leaf 노드인 경우의 비용은 그 노드의 traffic percentage로만 나타나지만, leaf 노드가 아닌 노드의 비용은 그 노드의 traffic percentage뿐만 아니라 자신의 하위 노드들의 비용도 함께 고려해야 한다.

주어진 트리 구조 모델에서 임의의 노드 i 에 대해 부모노드는 PA_i 라하고, 노드 i 의 자식 노드들의 집합을 CH_i 라하며 노드 i를 root로 하는 서브 트리에서 i를 제외한 노드의 집합은 SU_i로 정의하자. 노드 i 의 비용 중 자신의 traffic percentage만으로 인한 비용은 $\text{cost}(i) = T[i] - \sum_{j \in CH_i} T[j]$이고, 노드 i를 서브 트리의 루트로 했을 때 어느 하위 노드 $k(k \in SU_i)$의 비용을 $\text{cost}(i, k)$라고 하면 이 값은 다음과 같이 정의된다.

$$\text{cost}(i, k) = (P[k] - P[i]) \times (T[k]) + \sum_{s \in SU_k} \text{cost}(i, s) \quad (1)$$

예를 들면 그림 2에서 노드 i를 서브 루트로 했을 때 노드 6의 비용은 $\text{cost}(9, 6) = (P[6] - P[9]) \times (T[6])$ 이 되고, 리프 노드가 아닌 노드 4의 비용은 $\text{cost}(9, 4) = (P[4] - P[9]) \times (T[4]) + \text{cost}(9, 1) + \text{cost}(9, 2) + \text{cost}(9, 3)$ 이 된다.

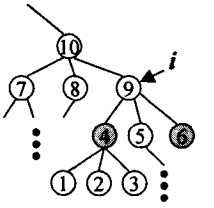

그림 2 트리 구조의 한 부분

주어진 문제의 해결을 위한 proxy의 개수와 위치를 찾는 알고리즘을 설명하기 위해 다음과 같은 표시법을 사용한다. V는 주어진 트리 구조의 노드들의 집합을 나타내고, 트리의 깊이를 d라 한다. $PROXY[i]$는 알고리즘의 결과로 계산되는 i번째 proxy의 위치를 나타내고 총 proxy의 개수를 나타내는 마지막 i 값을 M으로 표기한다. 또 Ts는 각각의 proxy server가 보장하는 지연 임계시간 (Threshold delay time)이다. 지연 임계시간 Ts는 최대 노드 비용 $\max_{i \in V}\{\text{cost}(i)\}$ 보다 항상 크거나 같다고 가정한다.

알고리즘은 먼저 트리의 최하위 레벨 d 에 있는 노드들로부터 레벨 1에 있는 root까지 레벨 단위로 수행이 되는데, 레벨 l에서 현재 작업중인 임의의 노드를 i라 할 때 노드 i와 그 서브 트리의 비용 $(\text{cost}(i) + \sum_{j \in Ch_i} \text{cost}(i, k))$을 구하여 SCh 에 저장하고 그 값이 임계값 Ts보다 크면 노드 i의 자식 노드들 중 가장 큰 비용을 갖는 노드 k 를 찾아 새로운 proxy로 정해주고, 노드 집합 V에서 노드 k 와 그의 서브 트리에 있는 모든 노드들을 제외시킨다. 나머지 노드들로 SCh 값을 재 계산 해주고 위의 과정을 SCh 값이 임계값 보다 작을 때까지 계속한다. 이러한 과정을 root까지 수행하게 된다. 알고리즘 1의 Line 1에 있는 while

```
< Algorithm 1>  Proxies_in_Tree
input :  V, Ts, P[i], T[i]
output :  PROXY[M], M
/* Initialization */
0:  M ← 0, Total ← 0, SCh ← 0, l ← d
/* algorithm */
1: while  (l ≥ 1)
2:   for each node i at level l
3:     SCh ← cost(i) + Σ_{k∈SU_i} cost(i,k)
4:     while SCh > Ts
5:       Mch ← j such that max_{j∈CH_i}{cost(i,j)}
6:       PROXY[M] ← Mch, M ← M+1
7:       SCh ← SCh − cost(i, Mch)
8:       V ← V − {{Mch} ∪ SU_Mch}
       endwhile
     endforeach
9:   l ← l−1
   endwhile
```

loop 이 끝났을 때 남는 노드들은 web server로부터 직접 서비스를 받게된다.

정리 1. Tree $T[V,E]$와 각 노드 i에 대한 $P[i]$, $T[i]$값과 임계 지연시간 Ts 가 주어졌을 때, 알고리즘 Proxies_in_Tree에 의해 찾아지는 M은 최소 값이 되며 각 proxy들의 cost는 임계값을 넘지 않는다.

증명 : 노드 i를 루트로 하는 서브 트리를 $T(i)$라 하고 $T(i)$의 총 비용을 $\text{cost}(T(i)) = \text{cost}(i) + \sum_{k\in Ch_i} \text{cost}(i,k)$라 하자. 그림 3에서 현재 고려중인 노드가 i이고 CH_i가 $\{i_1, i_2, \cdots, i_{|CH_i|}\}$ 일 때, 노드 i의 자식 노드들의 이름이 각각을 루트로 하는 서브 트리의, 비용들간의 관계를 $\text{cost}(T(i_1)) \geq \text{cost}(T(i_2)) \geq \cdots \geq \text{cost}(T(i_{|CH_i|}))$가 되게끔 정해졌다고 가정하자. 알고리즘 Proxies_in_Tree의 6번째 줄의 새로운 proxy에 선정되는 노드 j의 서브 트리의 총 비용은 항상 Ts를 넘지 않음을 볼 수 있다. 따라서 선택된 각각의 proxy가 자신의 클라이언트에게 서비스해야 하는 총비용은 Ts를 넘지 않는다. Proxies_in_Tree 알고리즘 4번째 줄의 while loop이 끝났을 때 남아있는 노드 i의 자식 노드들을 $i_{j+1}, i_{j+2}, \cdots, i_{|CH_i|}$라 하면 노드 i를 고려하여 생긴 proxy의 개수는 j개이고 남은 서브 트리의 총 비용은 $\text{cost}(i) + \sum_{j<k\leq|CH_i|} \text{cost}(i,i_k)$임을 알 수 있다. 이때, j 는 $\text{cost}(T(i))$의 값을 임계값 Ts 보다 작거나 같게 만들기 위해 삭제되어지는 노드의 최소 개수가 되고 동시에

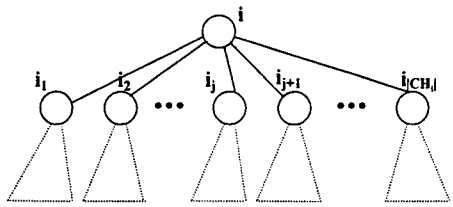

그림 3 서브 트리 T[i]

$\text{cost}(T(i_1)) \geq \text{cost}(T(i_2)) \geq \cdots \geq \text{cost}(T(i_{|CH_i|}))$ 조건에 의해 노드 i의 $|CH_i| - j$개의 자식노드로 만들 수 있는 서브 트리의 비용 중 최소값은 $\text{cost}(i) + \sum_{j<k\leq|CH_i|} \text{cost}(i,i_k)$이 됨을 쉽게 관찰 할 수 있다. 따라서 $T(i)$의 노드들은 $j+1$개 보다 적은 개수의 proxy들로 임계값을 넘지 않으면서 서비스를 받을 수 없으며 이러한 local의 최적값들의 선택은 global 최적값을 보장함을 쉽게 볼 수 있다. 따라서, Proxies_in_Tree 알고리즘이 구한 proxy의 개수 M은 주어진 조건을 만족하는 최소의 개수이다. ■

3. 결론

본 논문에서는 인터넷에서 트리 구조를 갖는 web server 시스템의 모든 proxy들이 주어진 임계시간을 초과하지 않으면서 클라이언트의 요구를 처리하는데 필요한 proxy의 최적 개수와 위치를 구하는 방법에 대한 알고리즘을 제안했다. 이는 시스템이 web proxy의 최적 개수를 구하고 그 proxy들을 적절한 곳에 배치함으로서 제한된 시간 안에 클라이언트의 요구를 보장해 주고, 시스템을 구축하고 유지하는 비용을 최소화하는데 그 의의가 있다. 앞으로 이 알고리즘을 확장하여 실제 인터넷 구조에 적용하는 연구가 필요하다.

참고

[1] N. Yeager and R. McGrath, Web Server Technology, Morgan Kaufman, 1996

[2] M. Baentsch, L. Baum, G. Molters. S. Rothkugel and P. Sturm, "World Wide Web Caching: The Application-Level View of the Internet." IEEE Communications Magazine, Vol.35, No.6, June 1997.

[3] D. Patterson and J. Hennessy, Computer Organization and Design: the Hardware/Software Interface, 2nd edition, Mogran Kaufman, 1997.

[4] B. Li, M. J. Golin and G. F. Italiano and X. Deng, "On the Optimal Placement of Web Proxies in the Internet: Linear Topology," In the 8th IFIP Conference on the High Performance Networking (HPN'98), Vienna, Austria, September 1998.

[5] B. Li, M. J. Golin and G. F. Italiano and X. Deng and K. Sohraby, "On the Optimal Placement of Web Proxies in the Internet," In IEEE InfoComm 1999, pages 1282-1290, 1999