분산 환경에서의 객체 캐싱

이태희 [1]　　심준호 [2]　　이상구 [3]

DOC: A Distributed Object Caching System for Information Infrastructure

Taehee Lee[1]　Junho Shim[2]　Sang-goo Lee[3]

Abstract

Object caching is a desirable feature to improve the both scalability and performance of distributed application systems for information infrastructure, the information management system leveraging the power of network computing. However, in order to exploit such benefits, we claim that the following problems: cache server placement, cache replacement, and cache synchronization, should be considered when designing any object cache system. We are under developing DOC: a Distributed Object Caching, as a part of building our information infrastructures. In this paper, we show how each problem is inter-related, and focus to highlight how we handle cache server deployment problem

Key Word *: Distributed Database, Object Caching, Web Caching, Server Placement*

[1] 서울대학교 컴퓨터공학과 박사과정　　thlee@europa.snu.ac.kr
[2] 숙명 여자 대학교 컴퓨터과학과 조교수　　jshim@sookmyung.ac.kr
[3] 서울대학교 컴퓨터공학과 부교수　　sglee@europa.snu.ac.kr

1. 서론

인터넷의 급속한 발전과 전자 상거래의 활성화로 인해 수많은 기업들이 온라인 상으로 사업영역을 확장하려 하고 있다. 이에 따라 지역적으로 멀리 떨어진 여러 종류의 정보 제공자들이 서로 다른 포맷으로 제공하는 정보들이 전자 상거래 애플리케이션에 사용되기 위해서 통합 관리 되어야 할 필요성이 대두 되었고, IBM, Oracle, Microsoft 등 대형 소프트웨어 개발 업체에서 J2EE 또는 COM+ 기반의 분산 환경에서의 정보 관리 시스템을 개발하고 있다.

이러한 분산 정보 관리 시스템에서 애플리케이션들의 성능과 확장성을 높이기 위해서 자주 쓰이는 방법이 데이터의 캐싱이다. 일반적인 분산 환경에서 각 사이트는 그 사이트 내의 데이터만을 사용하지 않고 네트웍상의 멀리 떨어진 다른 사이트의 데이터를 요청하여 사용하게 되는 일이 발생하게 되는데 이러한 경우 데이터의 전송 속도를 향상시키기 위하여 자주 쓰이는 데이터들을 클라이언트들이 밀집한 곳에 캐싱하여 사용하게 된다. 이러한 캐싱을 사용함으로써 데이터가 전송되는 시간을 줄일 수 있을 뿐 아니라, 로드 밸런싱 (load balancing), 장애 무결성 (fault-tolerance)에도 도움을 줄 수 있다[8, 11].

분산 애플리케이션 시스템에서 캐싱을 효율적으로 적용하기 위해서 다음과 같은 세가지 문제점이 해결되어야 한다. 첫 번째는 네트웍상의 어느 곳에 캐쉬를 두어야 하는가 하는 "Cache Deployment" 문제이다. 두번째는 제한된 크기를 갖는 캐쉬내에 모든 데이터들을 저장 할 수가 없기 때문에 어떤 데이터를 교체해야 하는가 하는 "Cache Replacement" 문제이다. 마지막으로 여러 캐쉬들과 데이터 원본을 가지고 있는 서버간의 무결성을 유지하는 "Cache Synchronization" 문제이다.

분산 환경에서 자주 쓰이거나 중요한 데이터는 데이터 센터로 지정된 몇몇의 사이트에 캐싱된다. Cache Deployment 알고리즘은 전체적인 시스템의 성능을 높이기 위하여 어떤 사이트를 데이터 센터로 지정할 것이가를 정하는 방법으로써 데이터의 사용 패턴에 따라 정적 또는 동적으로 데이터 센터의 위치를 정하게 된다. 일반적으로 클러스터링 방법을 이용하거나 휴리스틱을 사용하

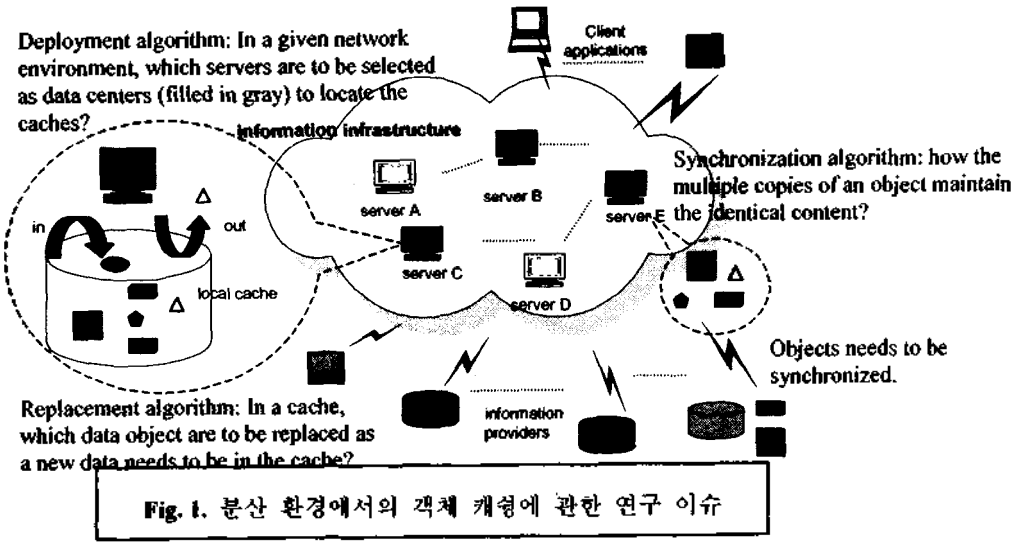

Fig. 1. 분산 환경에서의 객체 캐싱에 관한 연구 이슈

여 데이터 센터의 위치를 정하게 되는데, 본 논문에서는 파티셔닝 클러스터링 알고리즘[5]을 사용하여 데이터 센터의 위치를 찾아낸다.

Cache replacement 문제는 버퍼 관리[2]와 같은 분야에서 심도 깊게 연구된바 있는데, 캐쉬안에 보관할 데이터 아이템들을 동적으로 정하는 문제이다. 보통 미래에 참조될 가능성이 높은 데이터를 예측하여 캐쉬의 hit ratio 를 높이려는 방법이 제안되었으나, 분산 시스템에서는 hit ratio 를 최대화 하는 것 만으로는 클라이언트들이 느끼는 응답 시간을 최소화 할 수 없는 경우가 많이 있다. 이 경우에는 데이터가 캐쉬안에 없을 경우에 이 데이터를 서버로부터 가져오는데 필요한 응답 시간인 cache miss 비용을 최소화 할 수 있도록 디자인 되어야 한다.

분산 환경에서 데이터는 여러 개의 데이터 센터에 복제되어 캐쉬될 수 있다. 이 경우 애플리케이션에 의해 어느 한 데이터 센터 내에 캐쉬된 데이터가 변했을 경우 이러한 사실이 그 데이터를 가지고 있는 모든 캐쉬들에 전파되어 여러 사이트에서 진행되는 트랜잭션들의 처리에 문제가 발생하지 않도록 해야한다. Data synchronization 에서는 이런 데이터의 최신성을 보장할 수 있는 방법을 마련해야 한다.

본 논문에서는 I_2; *Information Infrastructure* 라는 이름으로 진행되었던 객체지향 환경에서 분산 애플리케이션들을 지원하는 정보 관리 시스템 개발 프로젝트에서 진행되었던 연구들 중 캐쉬과 관련된 부분에 대하여 정리하였다. 특히, Cache replacement 에 대해서는 간단히 설명하고 Cache deployment 문제에 대하여 알고리즘과 실험 결과를 포함하여 자세히 기술하였다. 본 논문의 구성은 다음과 같다. 2 장에서는 논문에서 제안한 cache deployment 알고리즘에 대하여 기술하고 4 장에서 cache replacement 와 관련된 이슈들을 정리하고 5 장에서 결론과 향후 연구 계획에 대해 정리하였다.

2. Cache Deployment 알고리즘

네트웍상에 캐쉬의 적절한 위치를 찾기 위한 여러 가지 목적 함수가 있을 수 있다. 예를 들어 지리적으로 가까운 클라이언트들끼리 클러스터를 생성하고, 각 클러스터의 무게 중심점을 찾아냄으로써 캐쉬 위치를 찾아낼 수도 있다. 본 논문에서는 네트웍 통신 속도를 고려하여 클라이언트와 데이터 센터간의 통신 지연 시간을 최소화하는 위치를 찾는 방법을 제안한다.

제안하는 방법에서는 분산 환경의 네트워크 구조를 그래프 형태로 모델링 한다.

> 네트웍 그래프 $G=(V,E)$, where V is the set of nodes and $E \subseteq V*V$, the set of links

그래프에서의 각 노드는 일관된 관리 방법과 routing 정책을 갖고 있는 AS (Autonomous System)을 나타내고, 각 링크는 각 노드간의 통신 대역폭 값을 갖고 있는 네트웍 연결을 나타낸다. 이러한 모델하에서 각 노드간의 통신 지연 속도는 , 다음의 식으로 구할 수 있다.

> $d_{ij} = \sum_{e \in p_{ij}}^{k} c \times \frac{1}{b_e}$, p_{ij}는 노드 i와 j 간의 최단 거리 링크이고 b_e는 통신 대역 값, c는 상수

네트웍상에 n 개의 클라이언트 $c_i \in C=\{c_1, c_2, ..., c_n\}$와 k 개의 서버 s_j, $1 \le j \le k$ 가 있을 때, S_j 를 로부터 데이터를 공급받는 클라이언트

의 집합이라 하고 다음과 같은 성질을 갖는다.

$$\bigcup_{j=1}^{k} S_j = C, S_l \cap S_m = \Phi, \text{ if } l \neq m.$$

이경우 cache deployment 알고리즘은 다음의 전체 네트웍 지연 시간을 최소화 할 수 있

$$D(C, S, k) = \sum_{j=1}^{k}\sum_{c_i \in S_j}(d_{ij} \cdot n_{ij}) = \sum_{j=1}^{k}\sum_{c_i \in S_j}\left(\sum_{e \in p_e} c \times \frac{1}{b_e} \cdot n_{ij}\right)$$

도록 디자인 되어야 한다.

여기서 d_{ij} 는 서버 s_j 로부터 클라이언트 $c_i \in S_j$까지의 평균 통신 지연 시간이고, n_{ij}은 데이터 요청양이다.

이러한 네트웍 지연 시간과 더불어 본 논문에서 제안하는 방법은 데이터 센터를 이전하였을 때 발생하는 비용에 대해서도 고려하였다. 즉, 기업 또는 서비스 관리 업체가 네트웍상의 임의의 위치에 새로운 데이터 센터를 추가하거나 또는 비용이나 정치적 이유로 인해 데이터 센터를 추가하지는 못하고 기존의 데이터 센터 중 일부의 위치를 변경할 수만 있는 경우가 생길 수 있다. 이 경우 전체 데이터 센터의 위치를 모두 변경하는 경우하는 것보다 일부의 위치만을 변경함으로써 이전 비용을 줄일 수 있다. 본 논문에서는 이러한 경우를 고려하여 새로 추가할 데이터 센터의 개수와 이전할 수 있는 데이터 센터의 개수가 제한되는 경우를 고려하여 다음과 같은 총 네트웍 비용식을 고안 하였다.

$$D_{reorg}(C, S, k, k', \delta) = \sum_{j=1}^{k'}\sum_{c_i \in S_j}(d_{ij} \cdot n_{ij}) + \sum_{j=k'+1}^{k}\sum_{c_i \in S_j}(d_{ij} \cdot n_{ij}) + \sum_{j=k+1}^{\delta+k}\sum_{c_i \in S_j}(d_{ij} \cdot$$

여기서 k 는 새로 추가될 데이터 센터를 포함한 데이터 센터의 개수, k'은 기존 데이터 센터 중 위치가 변하면 안되는 데이터 센터의 개수, δ는 새로 추가될 데이터 센터의 개수 이다.

지난 연구인 [10]에서 대표적인 파티셔닝 클러스터링 알고리즘인 CLARANS 에 기반하여 SEPA 라는 서버 위치 알고리즘을 개발하였다. SEPA 알고리즘은 계층적 클러스터링 알고리즘중 가장 성능이 좋다고 알려진 CURE 보다 더 좋은 성능을 보였었다. [13]에서는 SEPA 를 확장하여 위에서 제시한 그래프 형태의 네트웍 토폴로지에 대하여 실험을 하였고 그 성능이 [10]에서와 마찬가지로 CURE 나 Greedy 알고리즘 보다 우수함을 보였다. 그림 2 에 [13]에서 제안한 변형된 SEPA 알고리즘을 제시하고 있다.

SEPA algorithm

Input : the number of clusters k, the object set C with n objects c_i, *max_neighbor*, and *num_local*, the set of pre-computed clusters $S = \{S_1, S_2, \ldots, S_k\}$, k': the number of servers to be kept at current locations, and δ : the number of new servers to the network

Output : A set of k clusters $S = \{S_1, S_2, \ldots, S_{k+\delta}\}$ that minimizes

$$\sum_{j=1}^{k'}\sum_{c_i \in S_j}(d_{ij} \cdot n_{ij}) + \sum_{j=k'+1}^{k}\sum_{c_i \in S_j}(d_{ij} \cdot n_{ij}) + \sum_{j=k+1}^{\delta+k}\sum_{c_i \in S_j}(d_{ij} \cdot n_{ij})$$

, and their server locations $s_1, s_2, \ldots, s_{k+\delta}$

Method :

(1) repeat

① Set *num_no_change* to 0.

②Find k initial medoids m_i, $1 \le i \le k$. Randomly select δ more medoid objects and set them $m_{k+1}, m_{k+2}, \ldots, m_{k+\delta}$

③repeat

A.Assign each remaining object to the cluster with the nearest medoid. Assign c_i to the cluster S_i with the minimal d_{ij}. Compute the total cost $D = \sum_{j=1}^{k+\delta}\sum_{c_i \in S_j}(d_{ij} \cdot n_{ij})$.

④Randomly select a non medoid object, crandom.

⑤If swaping crandom with a medoid mi, k'+1≤i ≤k results in more than (k-k') changes of medoids, do not swap. Repeat B.

⑥Compute the total cost,

⑦

$D_{new}=$

$$\sum_{j=1}^{k'}\sum_{c_i\in S_j}(d_{ij}\cdot n_{ij})+\sum_{j=k'+1}^{k}\sum_{c_i\in S_j}(d_{ij}\cdot n_{ij})+\sum_{j=k+1}^{\delta+k}\sum_{c_i\in S_j}(d_{ij}\cdot n_{ij})$$

, of swapping m_i with c_{random} with a medoid m_i, $k'+1 \le i \le k$.

B. If $D_{new} < D$ then swap m_i with c_{random} to form the new set of k medoids.

C. Otherwise, increase *num_no_change* by 1.

⑧ Until *num_no_change* ≤ *max_neighbor* do ③.

⑨ Set S_OPT_{local} to the current set $S=\{S_1, S_2, \ldots, S_{k-\delta}\}$, increase *local* by 1.

⑩ Until *local* ≤ *num_local* do (2) to find out next local optimal set.

(2) Find a S from $S\_OPT_1, S\_OPT_2, \ldots, S\_OPT_{local}$, which results in the minimal cost. And return the medoids . m_i, $1 \le i \le k+\delta$ as center locations of clusters.

Fig. 2. Server Deployment algorithm : SEPA

3. Cache Replacement 알고리즘

본 논문에서 제안하고 있는 Cache replacement 알고리즘은 [11]에 기반하고 있다. 기존의 알고리즘들이 캐쉬 hit-ratio 만을 성능 지표로 삼고 있는데 반해 논문에서 제안하는 시스템의 경우 cache 에 데이터를 저장함으로써 생기는 cost saving ratio (CSR)을 사용한다.

> CSR = $\Sigma_{i\in I}\, h_i \cdot c_i / f_i \cdot c_i$, h_i 는 cache hits 수, f_i is 캐쉬 참조 수

임의의 데이터 i 의 비용(cost)은 클라이언트로부터 데이터가 요청된 후, 데이터 원본으로부터 캐쉬에 그 데이터가 도착하기까지의 turn around time 에 의해 측정한다. 이때 단 한번의 시간 측정으로 비용을 결정하지 않고, 과거 K 개의 시간 측정의 평균값을 사용하도록 한다. 클라이언트의 데이터 요청은 특정 Object ID (OID)를 요구하는 경우, 원격 함수 호출로 요청하는 경구, 또는 데이터 질의 언어를 사용하는 경우가 있을 수 있다. 첫번째 경우에는 주로 네트웍 전송 속도에 의해 turn around time 이 결정되지만 나머지 두가지 경우에는 서버 측의 처리 속도에따라 비용이 결정되기도 한다.

Cache replacement 알고리즘이 적용 가능성 여부는 알고리즘의 예측 성능 이외에도 알고리즘이 얼마나 빨리 캐쉬에서 추방되어야 할 데이터를 찾아내는 가도 중요한 결정 요소이다. 일반적으로 비용 기반 알고리즘들은 각 데이터별로 데이터의 비용을 메타 데이터로 보관하고 정렬하여 가지고 있다. 본 논문에서 구현한 cache replacement 알고리즘도 각 데이터 별로 데이터 크기, 이득값, OID, 과거 K 개의 turn-around time 을 갖고 있다. 이런 모델하에서 단순하게 구현할 경우 cache replacement 알고리즘은 캐쉬내에 속한 데이터들을 정렬하는데 드는 비용인 $O(n \cdot log\ n)$의 시간 복잡도를 갖는다. 그러나 본 논문에서 제안하는 방법은 데이터들이 완전하게 정렬될 필요없이 profit 값이 최소인 데이터만을 찾아내면 되므로 heap 구조를 사용하여 $O(log\ n)$의 시간 복잡도로 cache replacement 를 구현할 수 있다.

4. 결론

본 논문에서는 I_2 : *Information Infrastructure* 프로젝트의 결과물로써 [11]과 [13]에서 제안한 SEPA 알고리즘의 개요와 시스템에서 구현한 Cache replacement 알고리즘에 대해 기술하였다. SEPA 는 CLARANS, CURE, Simulated Annealing[5], Genetic Algorithm, Autoclass, CLARA 등 여러가지 알고리즘들을 구현한 결과 가장 성능이 좋았던 CLARANS 알고리즘에 기반하여 고안된 알고리즘이다. [13]에서 제시한 바와 같이 SEPA 는 물리적 거리 (Euclidean distance)에 기반한 네트웍 모델 뿐 아니라 네트웍 전송 대역을 고려한 그래프 형태의 네트웍 모델

에 대해서도 좋은 성능을 보였다. 향후 네트웍 혼잡도등 네트웍 성능에 영향을 미치는 몇가지 요인을 고려한 다른 모델에 대해서도 이 알고리즘의 적합성 여부를 실험할 계획이다.

웹 또는 분산 환경에서 캐쉬 기법을 도입함으로써 서버와 클라이언트 간의 통신 지연 시간의 최소화, 로드 밸런싱 (load balancing), 장애 무결성 (fault tolerance)등의 효과를 얻을 수 있다. 앞서 제시한 Cache deployment 문제는 첫번째 효과인 통신 지연 시간의 최소화에 큰 도움을 준다. 이밖에 Cache replacement 는 제한된 크기의 캐쉬 내에 저장할 가장 적합한 데이터를 찾음으로써 시스템의 성능 향상에, Cache synchronization 은 서버들에 복제되어 있는 데이터들간의 무결성 보장에 도움을 준다. 본 논문에서는 Cache deployment 와 replacement 에 있어서 현재까지의 연구 성과를 요약했으며 향후 synchronization 등의 다른 분야로 연구 분야를 확장할 계획이다.

참고문헌

1. K. Calvert, and E.Zegura. GT Internetwork Topology Models (GT-ITM). http://www.cc.gatech.edu/fac/Ellen.Zegura/gt-itm.

2. D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, 1991.

3. M. J. Franklin, M. J. Carey, and M. Livny, "Transactional Client-Server Cache Consistency: Alternatives and Performance", ACM Transactions on Database Systems 22(3), 1997.

4. S. Guha, R. Rastogi, and K. Shim, "CURE: An Efficient Clustering Algorithm for Large databases," Information Systems, Vol. 26, No. 1, Pergamon, 2001.

5. J. Han, and M. Kamber, Data Mining : Concepts and Techniques, Morgan Kaufmann, 2001.

6. S. Jamin, C. Jin, A. R.Kurc, D. Raz, and Y. Shavitt, "Constrained Mirror Placement on the Internet", Proc.of IEEE Infocom, 2001

7. K. Leung, J. Shim, D. Tcherevik, and A. Vinberg, "A Scalable Yet Transparent Infrastructure for Distributed Applications," Proc. of the 8th International Conference on Parallel and Distributed Systems, IEEE Computer Society, 2001.

8. P. Krishnan, D. Raz, and Y. Shavitt, "The Cache Location Problem," IEEE/ACM Transactions on Networking, Vol. 8, No. 5, ACM, 2000.

9. P. Radoslavov, R. Govindan, and D. Estrin, "Topology-informed internet replica placement", Proc. of WCW'01: Web Caching and Content Distribution Workshop, 2001

10. J. Shim, T. Lee, and S. Lee, "A Server Placement Algorithm Conscious of Communication Delays and Relocation Costs", Web Engineering '2002 / Lecture Notes in Computer Science, Vol. 2376, Springer-Verlag, 2002

11. J. Shim, P. Scheuermann, and R. Vingralek, "Proxy Cache Algorithms: Design, Implementation, and Performance," IEEE Transactions on Knowledge and Data Engineering, Vol. 11, No. 4, IEEE Computer Society, 1999.

12. L. Qiu, V.N. Padmanabhan, and G.M. Voelker, "On the Placement of Web Server Replicas," IEEE INFOCOMM, 2001

13. Taehee Lee, Junho Shim, Sang-goo Lee, "DOD : A Distributed Object Caching System for Information Infrastructure," HIS 2003/ Lecture Notes in Computer Science, Vol 2713, Springer Verlag, 2003