구조화된 동적 P2P시스템에서의 로드 변화 예측을 통한 효율적인 로드 밸런싱

최연오° 송진우 양성봉
연세대학교 컴퓨터과학과
{yeonoz°, fantaros, yang}@cs.yonsei.ac.kr

Effective Load Balancing through Predicting Load Change in Dynamic Structured P2P Systems

YeonOh Choi°, JinWoo Song, SungBong Yang
Dept. of Computer science, Yonsei University

요 약

본 논문에서는 분산 해쉬 테이블(Distributed Hash Table)을 이용하는 구조적 P2P(Peer-to-Peer) 시스템에서의 로드 밸런싱 문제를 다룬다. 이러한 환경에서 시스템은 여러 가지 이유로 각 피어들이 서로 다른 부하(load)를 갖는 상황을 맞이하게 된다. P2P 시스템에 오브젝트가 지속적으로 삽입되고 삭제되는 동적인 환경에서 오브젝트의 생존시간을 고려하여 피어가 갖는 로드의 변화를 예측함으로써 불필요한 로드의 이동을 제거하는 효율적인 로드 밸런싱 기법을 제안한다.

1. 서 론

본 논문에서는 DHT(Distributed Hash Table)를 이용하는 구조적 P2P(Peer-to-Peer) 시스템에서의 로드 밸런싱 문제를 다룬다([1],[2]). DHT P2P 시스템에서 각 오브젝트와 피어는 해쉬 함수에 의해 고유한 식별자(Identifier)를 갖게 된다. 이러한 환경에서 시스템은 여러 가지 이유로 각 피어들이 서로 다른 로드(load)를 갖는 상황을 맞이하게 된다. 첫 번째 이유로 각 피어와 오브젝트의 식별자가 시스템이 수용하는 ID 공간에서 불균일함을 들 수 있다[3]. 둘 중 하나가 균일하더라도 나머지가 불균일하다면 피어의 로드가 불균형을 이룬다. 두 번째로 시스템을 이루는 각 피어의 능력의 상이함을 들 수 있다[3]. 피어와 오브젝트의 식별자가 균일하다 하더라도 각 피어의 능력(저장 공간이나 네트워크 대역 등)이 다르다면 시스템이 불균형을 이룬다.

기본적으로 로드 밸런싱 알고리즘은 다음과 같은 목적을 달성하기 위해 연구된다.

1) *로드의 불균형을 최소화 한다.*
2) *알고리즘 수행 시 로드의 이동을 최소화 한다.*

본 논문에서는 1)의 측면에서는 기존 알고리즘과 비슷한 성능을 보이지만 2)의 측면에서의 성능향상을 목표로 하는 알고리즘을 제안한다. 오브젝트가 계속적으로 삽입되고 삭제되는 동적 환경에서 로드의 생존시간을 고려하여 시스템을 구성하는 피어의 로드의 변화를 예측함으로써 로드 밸런싱 수행 시 불필요한 로드의 이동을 제거할 수 있다.

2. 관련 연구

현재 몇 가지 로드 밸런싱 해결방안이 제안되어 있지만 대부분 정적인 시스템 환경을 가정하고 있다([1],[3],[4]). 일부는 동적인 환경에서의 로드 밸런싱을 다루기도 하였다[5]. 본 논문 역시 [5]와 같이 오브젝트가 피어의 실제 저장 공간에 지속적으로 저장되고 삭제되는 환경을 가정하지만 피어의 참여와 탈퇴는 없는 것으로 가정한다.

P2P 로드 밸런싱에는 크게 두 가지 방법이 있다. 첫 번째로 가상 서버(virtual server)개념을 로드 밸런싱에 응용한 연구가 있다[6]. DHT 환경에서 오브젝트 키의 저장과 라우팅은 물리적 피어 레벨이 아닌 가상 서버 레벨에서 이루어진다. 물리적 피어는 하나 이상의 가상 서버를 가질 수 있으며 로드 밸런싱은 가상 서버들을 무거운 물리적 피어로부터 가벼운 물리적 피어로 옮김으로써 수행된다.

두 번째 로드 밸런싱 방법으로 "*Power of two choices*"패러다임을 이용한 연구가 있다[4]. 각 오브젝트는 2개 이상의 해쉬 함수에 의해 식별자를 가지며 그 식별자들을 책임지는 각각의 피어 중 가장 적은 로드를 가지는 피어에 오브젝트의 키가 저장된다. 나머지 피어는 실제로 키를 저장하고 있는 피어의 링크를 유지함으로써 검색을 가능하게 하는 방식이다.

본 논문에서는 이미 제안된 P2P 로드 밸런싱 기법 중 가상 서버 기법을 이용한다.

3. Load Balancing 알고리즘

3.1 로드 예측

일반적으로 클라이언트가 가지는 로드는 그것이 담당하는 키의 수에 비례한다고 볼 수 있다. 로드의 양이 변한다는 것은 가지고 있는 키의 양이 변함을 의미한다. P2P 환경에서 클라이언트가 담당해야 하는 키의 발생과 사라짐은 해당키가 가리키고 있는 실제 오브젝트가 발생하고 사라지는 것을 의미한다. 따라서 클라이언트의 로드는 그 클라이언트가 가지는 키들에 해당하는 오브젝트들의 생존시간(*Life Time*)을 이용하여 예측할 수 있다.

오브젝트의 생존시간은 그 오브젝트를 실제로 저장하고 있는 피어에 따라 다르게 예측될 수 있다. LT_A를 피어 A에 저장되어 있는 오브젝트의 생존시간이라고 한다면 LT_A는 식 (1)과 같이 피어 A의 평균적인 오브젝트 교체시간으로 예측할 수 있다. 식 (1)에서 TR과 TS은 각각 오브젝트가 삭제되고 저장된 시각이고 q_A는 피어 A에서 삭제된 오브젝트의 수이다.

$$LT_A = \frac{\sum_{i=1}^{q_A}(TR_i - TS_i)}{q_A} \quad (1)$$

따라서 DT^t_A를 시각 t에 피어 A에 저장된 오브젝트의 사라질 시각이라고 한다면 이를 식 (2)와 같이 예측할 수 있다.

$$DT^t_A = t + LT_A \quad (2)$$

3.2 OLAB(Object Lifetime Aware Load Balancing)

이 절에서는 각 피어의 로드의 변화를 고려한 OLAB(Object Lifetime Aware Load Balancing)이라고 부를 로드 밸런싱 알고리즘을 제안한다. 제안하는 알고리즘은 기본적으로 무거운 로드를 가진 피어가 가벼운 로드를 가진 피어에게 가지고 있는 가상 서버 중 일부를 이동시키는 *가상 서버*를 이용한 방식으로 동작한다[3].

3.2.1 가벼워질 피어

L_A와 C_A를 각각 피어 A의 로드와 수용능력이라 한다면 $L_A > C_A$일 때 피어 A를 무겁다고 하고 그렇지 않을 때 가볍다고 정의한다[3]. 오브젝트에 대한 생존 시간을 예측할 수 있을 때 어떤 오브젝트 즉 키는 현재시간을 기준으로 얼마만큼의 남은 생존시간을 가지는지 예측할 수 있다. RT_t와 ct를 각각 피어 A가 저장하고 있는 오브젝트의 남은 생존시간과 현재시각이라고 하면 이를 식 (3)과 같이 표현할 수 있다.

$$RT_t = DT^t_A - ct \quad (3)$$

이 때 TH_LEAVING_TIME이라는 한계 값(threshold)을 미리 정해놓고 $RT_t <$ TH_LEAVING_TIME일 때 해당 오브젝트를 *사라질 오브젝트*로 정의한다. DL_A를 피어 A가 갖는 곧 사라질 오브젝트 키의 수라고 하면 $L_A - DL_A \le C_A$ 일 때 피어 A를 *가벼워질 피어*로 정의한다.

3.2.2 가상 서버의 이동

로드 밸런싱의 가장 기본적인 동작은 가상 서버의 이동이다. 어떤 무거운 피어 h와 가벼운 피어 l사이에서 가상 서버 이동을 고려할 때 다음과 같은 원칙을 준수하여 *가장 알맞은 가상 서버* v를 선택하여 이동시키게 된다.

1. h에서 l로 v를 옮길 때 l을 무거운 피어로 만들지 않아야 한다.
2. v는 h를 *가벼워질 피어*로 만들 수 있는 가장 가벼운 가상 서버이어야 한다.
3. 2를 만족시킬 가상 서버가 없다면 가지고 있는 키들의 평균적인 생존시간이 가장 긴 가상 서버 v를 선택한다.

위의 원칙들은 근본적으로 가장 적은 양의 로드를 옮김으로써 무거운 피어 h를 가벼운 노드로 만들면서 가벼운 노드 l은 여전히 가벼운 피어로 남게 하고 있다. [3],[5]와의 주요 차이점이자 본 논문의 핵심은 원칙 2와 3에 있다. [3]에서 제시된 알고리즘과 비교했을 때 키의 생존시간이라는 요소를 고려하여 무거운 피어를 가벼워질 정도로만 만들어 줌으로써 곧 사라질 양의 로드만큼을 덜 이동시키는 효율적인 측면에서의 향상 가능성을 내포한다. 또한 성능적인 측면에서 로드 밸런싱을 수행한 직후에는 h가 여전히 무거운 피어일 수 있지만 일정시간 후에는 가벼워지거나 가벼운 피어에 가까워질 수 있는 피어로 되는 효과를 발휘한다.

본 논문에서는 위와 같은 알고리즘을 하나의 무거운 피어가 여러 가벼운 피어들을 고려하여 로드의 균형을 잡는 *일대다 방식*으로 구현하였다[3]. 일대다 방식에서 P2P 오버레이 네트워크상에 d개의 디렉토리가 존재하고 각 피어는 주기적으로 임의의 디렉토리에 자신의 로드상황을 알린다. 각 피어는 주기적으로 무거워질 경우 임의의 디렉토리에 등록되어 있는 가벼운 피어에게 로드를 분담시키게 된다. 디렉토리 D가 무거운 피어 B의 로드를 분배하는 알고리즘을 정리하면 다음과 같다.

```
for( D에 등록된 각 가벼운 피어 l에 대해)
    if( B는 가벼워질 피어)
        break;
    for( B의 각 가상 서버 v에 대해)
        if(load( v) + load( l) < capacity( l))
            가장 가벼운 v를 찾는다
    if( v를 찾을 수 없다면)
        키들의 평균 생존시간이 가장 긴 v를 찾는다
    v를 l로 이동시킨다
```

4. 실험 및 분석

4.1 측정 기준

본 논문에서는 두 가지의 주요 기준을 이용하여 실험 결과를 평가하였다.

1) *이동된 로드* : 1회의 로드 밸런싱 수행 시 이동된 가상 서버의 로드 총합이다. 로드 밸런싱에 드는 비용은 이동된 로드에 비례한다. 알고리즘의 효율적인 면을 측정하기위해 필요한 기준이다. 이 값이 작을수록 효율적인 알고리즘이라고 할 수 있다.

2) *99.9 피어 활용도* : 실험을 수행하는 시간동안 임의의 시각 t에 모든 피어의 활용도를 측정해서 가장 높은 값을 가지는 피어의 활용도이다. 이 값이 작을수록 로드 밸런싱이 잘 되었다고 할 수 있다. 피어 A의 활용도는 L_A/C_A이다.

모든 값은 시스템이 안정적인 상태일 때를 반영하기위해 몇 번의 로드 밸런싱 수행 후부터 측정하였다. 결과 그래프상의 값은 3번의 실험에 대한 평균값이다.

4.2 실험 환경

실험은 기본적으로 표 1과 같은 기본 값을 가지고 이루어졌다. 실험 목적에 따라 일부 요소를 변화시키면서 실험하였다.

표 1. 실험 환경 및 알고리즘 인자

환경 인자	기본 값
평균 시스템 활용도	0.8
오브젝트 발생간격	평균 0.001초 간격의 Poisson분포
오브젝트 발생 ID	ID 스페이스에 대해 균일 분포
피어의 수	4096
피어의 수용능력	shape 2, scale 38의 Pareto분포
피어의 저장공간	shape 2, scale 30의 Pareto분포

알고리즘 인자	기본 값
로드 밸런싱 주기	50초
피어가 소유한 가상 서버의 수	12
디렉토리의 수	10
TH_LEAVING_TIME	10초

4.3 실험 결과

그림 1과 같이 99.9 피어 활용도는 제안한 알고리즘이 오브젝트의 생존시간을 고려하지 않은 알고리즘 보다 로드 밸런싱 직후에는 다소 높지만 다음번 로드 밸런싱 전의 어느 시점부터는 계속해서 시스템이 더 좋은 상태에 있게 된다. 이는 알고리즘 수행 시에 무거운 피어의 로드 중 금방 사라질 로드는 이동시키지 않기 때문에 당시의 99.9 피어 활용도는 높지만 일정시간 경과 후엔 피어들이 비교적 비슷한 수준의 로드를 갖기 때문에 오브젝트가 지속적으로 시스템에 추가되더라도 작은 편차로 로드가 증가하게 되기 때문이다.

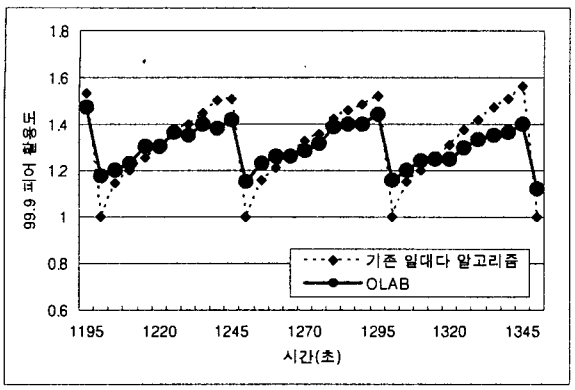

그림 1. 시간흐름에 따른 99.9 피어 활용도

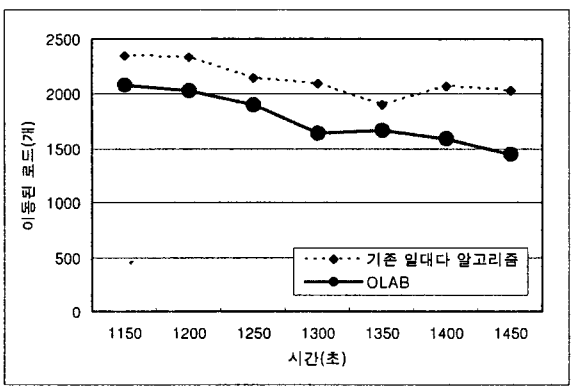

그림 2. 시간흐름에 따른 이동된 로드

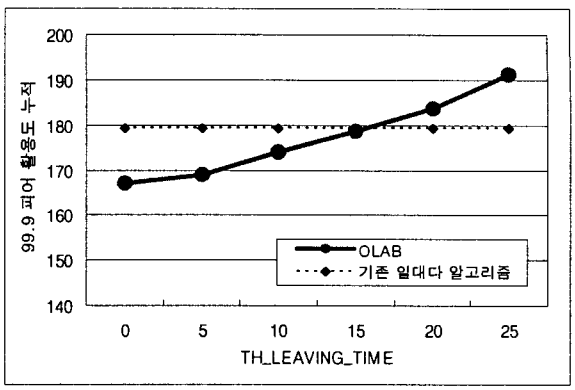

그림 3. TH_LEAVING_TIME의 변화에 따른 99.9 피어 활용도 누적

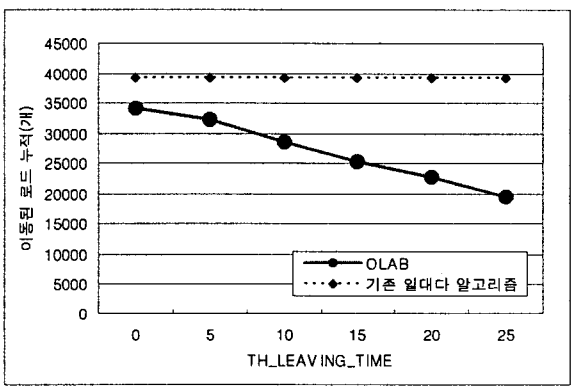

그림 4. TH_LEAVING_TIME의 변화 따른 이동된 로드 누적

그림 2는 실험을 수행하면서 발생되는 로드의 이동을 나타내는데 기존 알고리즘보다 25%정도의 향상이 있었다.

그림 3,4는 **TH_LEAVING_TIME**의 0부터 25까지 변화에 따른 99.9 피어 활용도와 이동된 로드를 보여준다. 그래프상의 값은 시스템이 안정 상태에 있는 실험시작 후 800초에서 1500초 사이의 누적 값이다. 그림 3에서 **TH_LEAVING_TIME**값을 작게 설정할수록 로드 밸런싱이 잘 되고 그림 4에서 **TH_LEAVING_TIME**값을 크게 설정할수록 더욱 효율적으로 로드 밸런싱을 수행하고 있음을 볼 수 있다. **TH_LEAVING_TIME**값이 크면 더욱 많은 로드를 금방 사라질 것으로 인정하기 때문에 로드 밸런싱 시에 그만큼의 로드를 옮기지는 않지만 무거운 피어를 가볍게 해주는 작용이 그만큼 약해지기 때문이다. 이와 같이 99.9 피어 활용도와 이동된 로드는 반비례 관계에 있다.

5. 결 론

본 논문에서는 P2P 시스템에 존재하는 오브젝트의 생존시간을 고려하여 피어가 갖는 로드의 변화를 예측함으로써 불필요한 로드의 이동을 제거하는 효율적인 로드 밸런싱 기법을 제안하였다. 제안한 알고리즘은 오브젝트의 생존시간을 고려하지 않은 알고리즘과 비교했을 때 99.9 피어 활용도 측면에서는 비슷한 수준이었으며 이동된 로드 측면에서는 알고리즘 인자의 변화에 따라 13%에서 30%정도의 성능향상이 있었다. 향후 P2P 네트워크상에서 지속적으로 피어가 참여하고 탈퇴하는 환경으로 알고리즘을 확장할 계획이다.

6. 참고문헌

[1] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," Proc. ACM SIGCOMM, pp.149-160, 2001.

[2] Antony Rowstron and Peter Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-Peer Systems," Proc. International Conference on Distributed Systems Platforms, pp.329-350, 2001.

[3] Ananth Rao, Karthik Lakshminarayanan, Sonesh Surana, Richard Karp, and Ion Stoica, "Load Balancing in Structured P2P Systems," Proc. Second Int'l Workshop Peer-to-Peer Systems (IPTPS), pp.68-79, 2003.

[4] John Byers, Jeffrey Considine, and Michael Mitzenmacher, "Simple Load Balancing for Distributed Hash Tables," Proc. Second Int'l Workshop Peer-to-Peer Systems (IPTPS), pp.80-87, 2003.

[5] Brighten Godfrey, Karthik Lakshminarayanan, Sonesh Surana, Richard Karp, Ion Stoica, "Load Balancing in Dynamic Structured P2P Systems," Proc. IEEE INFOCOM, Vol. 4, pp.2253-2262, 2004.

[6] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica, "Wide-area Cooperative Storage with CFS," Proc. 18th ACM Symp. Operating Systems Principles (SOSP), pp.202-215, 2001.