DHT 기반 P2P 시스템을 위한 적응적 근접 경로 선택 기법

송지영, 박성용
서강대학교 컴퓨터학과
itsuki@dcclab.sogang.ac.kr, parksy@sogang.ac.kr

An Adaptive Proximity Route Selection Scheme in DHT-Based Peer-to-Peer Systems

Ji-Young Song, Sung-Yong Park
Dept. of Computer Science, Sogang University

요 약

피어-투-피어 시스템에서는 한 홉(hop) 떨어진 두 피어간의 네트워크 상 거리는, 이들 피어들이 서로 다른 대륙간 네트워크에 존재할 수도 있기 때문에 매우 클 수 있다. 그러나 검색 질의 메시지를 전달할 다음 피어를 선택하는 기존의 방법은 네트워크의 근접성을 고려하지 않고, 단순히 목적 피어까지의 홉 수를 최대로 줄이는 피어를 선택한다. 따라서 네트워크의 근접성을 고려하여 동적으로 다음 라우팅 피어를 선택하는 새로운 기법이 필요하다. 본 논문에서는 대표적 DHT 방식 시스템인 Chord를 중심으로, 검색 시간을 향상시키기 위한 적응적 근접 경로 선택 기법을 제시한다. 본 기법에서는 동적인 P2P 환경에 적응하기 위해서 지수적 최근-가중치 평균을 통해 예측된 지연시간을 바탕으로 쿼리를 전달할 다음 피어를 선택한다.

1. 서 론

구조적 피어-투-피어 시스템은 안정한 오버레이(overlay) 네트워크 상에서 높은 확률로 검색의 성공을 보장할 수 있다[1]. 구조적 피어-투-피어 시스템은 주로 분산 해쉬 테이블(distributed hash table, DHT)을 사용한 검색 알고리즘을 이용한다. DHT 시스템은 오버레이 네트워크 상에서 <key,data> 쌍에 대한 확장적인 저장 및 검색을 위한 라우팅 인프라를 제공한다. 이들은 DHT를 사용해서 전체 식별자 공간(identifier space)(혹은 키 공간(key space))의 부분을 네트워크 상의 각 노드들에 할당하고 해당 노드들이 각각 할당된 키들에 대한 데이터를 관리하게 한다. 데이터에 대한 검색은 질의 메시지를 중간 노드들이 해당 목적 노드로 라우팅(routing)하는 방식으로 이루어진다.

DHT 방식의 피어-투-피어 시스템은 홉 카운트를 기준으로 효율적인 라우팅 방법을 제공한다. 대부분의 DHT 피어-투-피어 시스템은 평균적으로 O(log N)(N은 시스템 내의 노드의 수)의 어플리케이션 레벨의 홉(hop)간 메시지 전달을 통해 검색이 이루어질 수 있도록 한다. DHT 방식의 검색 프로토콜에서는 일반적으로 한 어플리케이션 홉을 진행할 때마다 해당 노드의 식별자가 찾고자 하는 데이터의 식별자 사이의 식별자 공간이 최소한 절반으로 줄기 때문이다. 그러나 오버레이 네트워크 상에서 임의의 두 노드간 라우팅 지연시간이 실제 기반 네트워크(underlying network)상의 유니케스트(unicast) 지연시간과는 다를 수 있다[2]. 예를 들어, 오버레이 네트워크 상에서 한 홉이 떨어진 거리는 실제 기반 네트워크 상에서는 서울에서 미국으로 라우팅이 될 수도 있고, 서울에서 부산으로의 라우팅이 될 수도 있다. 이것은 오버레이 네트워크의 위상과 실제 기반 네트워크의 위상은 일치하지 않기 때문이다. 어플리케이션 레벨 (오버레이 레벨)의 홉 카운트만을 최소화하도록 라우팅 피어를 선택하는 기존의 방법으로는 실제적인 검색 시간 향상을 기대하기 어렵다. 따라서 질의(혹은 쿼리)를 전달할 다음 홉의 피어를 선택할 때 홉 카운트뿐만 아니라 실제적인 근접성까지 고려하도록 하는 새로운 피어 선택 방법이 필요하다.

본 논문에서는 네트워크의 근접성을 고려하지 않은 기존 DHT 시스템의 검색 방법의 효율성 문제를 해결하고자, 검색 질의를 라우팅 할 때 데이터의 식별자에 대하여 이웃 피어를 좀 더 효과적으로 선택하기 위한 기법을 제시하였다.

본 논문의 나머지 구성은 다음과 같다. 2절에서는 본 논문의 피어 선택의 알고리즘을 구체적으로 제시한다. 3절에서는 실험을 통해 성능을 평가하고 마지막으로 4절에서는 결론을 내린다.

2. 지연시간 추정을 통한 라우팅 피어 선택법

2.1 지수적 최근-가중치 평균

광대역 환경과 같은 분산 환경은 매우 유동적이다. 많은 피어들이 새롭게 피어-투-피어 네트워크에 참여하거나, 예기치 못한 에러나 혹은 특정 이유를 목적으로 네트워크를 떠나곤 한다. 또한 오버레이 네트워크 상의 피어들 간의 지연시간도 시간에 따라서 동적으로 변할 수 있다. 이러한 환경에서 지연시간을 예측하기 위해서는 먼 과거 보다는 최근에 가중치를 두어 지연시간 비용을 예측하는 것이 타당하다. 이러한 일을 하는데 있어 일반적으로 잘 알려진 방법으로 지수적 최근-가중치 평균이 있다[4].

지수적 최근-가중치 평균의 특징은 현재의 값에 가장 큰 비중을 두면서도, 동시에 과거 측정값에 대해서도 지수적인 가중치를 두어 고려한다는 점이다. 즉, 현재 측정된 값이 급격히 증가하거나 급격히 감소할 경우에도 이에 따라 예측 값이 급변하지 않는 특성을 지니고 있다. 이러한 특성은 동적으로 변하는 링크의 특성을 반영하면서도 동시에 어플리케이션 홉이 지니고 있는 기본적인 링크 특성 역시 반영할 수 있게 해준다. 즉, 어플리케이션 레벨의 홉이 광역 링크(wide-area link)를 포함하고 있는지에 따라서 발생하는 차이를 지수적 최근-가중치 평균의 이러한 특징을 통해 잘 예측할 수 있다.

다음은 본 기법에서 사용하는 지수적 최근-가중치 평균의

일반적인 경우에 해당되는 식을 나타낸다. $Q_{n,k}$ 는 노드 n에서 k번째 추정 값을 의미한다.

$$Q_{n,k} = \alpha \cdot cost_k + (1-\eta) \cdot Q_{n,k-1}$$
$$= \alpha \cdot cost_k + (1-\alpha) \cdot \alpha \cdot cost_{k-1} + (1-\alpha)^2 \cdot Q_{n,k-2}$$
$$= \alpha \cdot cost_k + (1-\alpha) \cdot \eta \cdot cost_{k-1} + (1-\alpha)^2 \cdot \alpha \cdot cost_{k-2} + (1-\alpha)^3 \cdot Q_{n,k-3}$$
$$= \alpha \cdot cost_k + (1-\alpha) \cdot \alpha \cdot cost_{k-1} + (1-\alpha)^2 \cdot \alpha \cdot cost_{k-2} + (1-\alpha)^3 \cdot \alpha \cdot cost_{k-3}$$
$$+ ... + (1-\alpha)^{k-1} \cdot \alpha \cdot cost_1 + (1-\alpha)^k Q_{n,0}$$
$$= (1-\alpha)^k Q_{n,0} + \sum_{i=1}^{k} \alpha \cdot (1-\alpha)^{k-i} cost_i$$

여기서, 마지막 식의 계수의 합이 $(1-\alpha)^k + \sum_{i=1}^{k} \alpha \cdot (1-\alpha)^{k-i} = 1$ 이 되어 계수가 가중치 역할을 함을 알 수 있다. $\alpha \cdot (1-\alpha)^{k-i}$ 에 의해서 현재 k에서 cost가 측정된 시점 i까지 얼마나 떨어져 있는지 k-i에 따라서 가중치가 결정된다. α 가 $0 < \alpha \leq 1$ 사이의 값이라고 할 때 $1-\alpha$ 의 값은 1보다 작게 되므로 i시점에서 측정된 cost값의 비중은 k 가 증가함에 따라 지수적으로 감소하게 되는 것이다.

본 논문에서 쿼리 식별자에 따른 지연시간 예측을 위해 제안된 지연시간 업데이트 규칙과 지수적 최근-가중치 평균은 다음과 같다.

$$Q_n^{new}(i, nexthop(n)) = (1-\alpha) Q_n^{old}(i, nexthop(n))$$
$$+ \alpha \cdot (observed\ latency\ to\ nexthop(n) + \min_{z \in nexthops\ of\ nexthop(n)} Q_{nexthop(n)}(i, z))$$

여기서 $Q_n^{new}(i, nexthop(n))$ 는 피어 n이 식별자 i를 갖는 질의 메시지를 받았을 때, 이에 대해 이웃 노드 $nexthop(n)$ 을 선택했을 경우, i의 상속자에게 메시지가 전달되기까지 남은 예측된 지연시간을 의미한다. 각 피어는 질의를 받을 때 이 값들에 의해 다음 피어를 선택하게 된다. 이때 노드 n은 선택된 피어 $nexthop(n)$ 으로부터 $nexthop(n)$ 이 예측한 값을 얻어 이를 위해 (식 3)에 의해 갱신한다. 보통 스텝-크기(step-size) α 를 시간이 갈수록 작게 취해 최근에 얻어진 값보다 과거 값들에 더 비중을 두려고 하지만, 피어-투-피어 시스템과 같은 동적이고, 안정되지 않은 환경(non-stationary)에서는 이러한 방법은 효과적이지 못하다. 즉, 시간에 따라 변화하는 환경에는 오랜 과거의 결과가 현재에 미치는 영향이 상대적으로 작은 반면, 비교적 최근 결과의 영향이 더 중요하기 때문이다. 본 기법에서는 스텝-크기로 0.5의 상수 값을 취하여 과거의 값과 현재 값 사이의 오차를 같은 비율로 적용하여 새로운 값을 추정하고자 한다.

2.2 경로 선택 알고리즘의 설계

첫째, 질의 메시지를 받은 피어는 질의 메시지의 식별자에 따라 질의 메시지의 목적지 범위를 판단한다. 모든 쿼리 식별자에 대해서 지연시간 정보를 유지하면 보다 정확한 예측이 가능하지만, 이것은 많은 메모리 공간을 필요로 한다. 따라서 다음과 같이 쿼리의 식별자 범위를 결정하고자 한다. 현재의 피어 식별자는 n이고, 질의의 식별자가 id일 때,

$id \in [n + 2^i, n + 2^{i+1})$ (단, i는 $0 \leq i \leq m-1$ 사이의 자연수 일 때, m이 식별자의 bit 수)

이면 쿼리는 피어의 i번째 영역에 속하게 되고, 이때 이것을 쿼리 id는 i번째 상태에 있다고 정의한다. 각 피어에는 질의의 식별자에 따라서 총 m개의 쿼리 상태가 존재한다.

두 번째로, 각 피어는 식별자에 대해서 다음 라우팅의 후보가 되는 이웃 피어를 결정한다. 이때 Chord의 라우팅 규칙을 위반하지 않고 라우팅 될 수 있는 이웃 피어는 다음과 같다. 질의 식별자 id에 대해서 $nexthop(n) \in (n, id]$ 의 식별자를 갖는 모든 피어가 된다.

세 번째로, 위에서 얻은 각 이웃 피어들에 대해서 i상태의 쿼리를 보냈을 때 id의 상속자 피어까지의 추정-지연시간이 가장 작은 피어를 선택한다. 그러나 지연시간 추정의 방법과 네트워크의 동적 특성상 이 추정 값에는 에러가 있을 수 있기 때문에 가장 작게 추정된 피어와 원래의 Chord 검색 알고리즘이 선택하는 피어(질의의 식별자와 가장 가까운 이웃 피어)와의 추정 값의 차이가 ε 보다 크지 않으면 원래의 피어를 선택하는 휴리스틱을 생각해 볼 수 있다.

마지막으로, 선택된 이웃 피어에게 질의 메시지를 전달하고, 이웃 피어로부터 ACK 메시지를 받아 새롭게 추정된 지연시간을 받아 이 값을 앞에서 언급한 업데이트 규칙에 따라 갱신한다. 추정-지연시간에 대한 업데이트는 이렇게 자신이 선택한 이웃 피어에 대해 선택한 후에 이뤄질 수도 있지만, 좀더 빠른 추정을 위해 이웃 피어들로부터 주기적으로 업데이트 받을 수 있다.(예를 들면, Chord의 안정화 프로토콜 수행시)

```
// 노드 n은 식별자 r->x에 대한 검색 요청 r 을 처리한다.
processRequest(Node n, Request r)

 // 만약 n이 r->x의 상속자의 바로 앞 선임자라면
 if r->x exists between (n->id, successor(n)]
  // 요청 r을 생성한 노드에 successor(n)의 라우팅 정보를 전달한다.
  sendSuccessor(n, r->initiator, successor)
 // 그렇지 않으면,
 else
  // 식별자 r->x에 대한 범위 i를 구한다.
  i = getIDRange(n, r->x)
  // i 에 대해서 가장 작은 추정-지연시간을 갖는 이웃 피어 min_n을 찾는다.
  min_n = findMinEstimatedLatencyNeighbor(n->fingerList, i)
  // min_n에게 요청 r를 전달하고, 새롭게 측정된 추정-지연시간을 얻는다.
  new_estimated_latency = sendRequest(n, min_n, r)
  // 새로운 추정-지연시간으로 업데이트 규칙에 따라 테이블을 갱신한다.
  updateEstimatedLatencyTable(n->fingerList, i, new_estimated_latency)
```

그림 1 근접 경로 선택 기법의 의사 코드

2.3 라우팅 테이블 관리

피어-투-피어 시스템은 기존 피어가 시스템에 나가거나, 새로운 노드가 추가 될 수 있다. 뿐만 아니라, 각 피어의 라우팅 테이블의 크기가 제한되어 있으므로 한 피어에서 유지할 수 있는 이웃 피어의 수는 제한되어 있다. 각 피어는 자신의 라우팅 테이블을 이러한 조건에 따라서 관리해야 한다.

실제적인 Chord의 구현의 경우 확장성 있는 검색을 위해 기본적으로 유지하는 핑거 테이블 엔트리 노드들뿐만 아니라, 식별자 공간상에서 자신의 식별자 바로 다음으로 큰 식별자를 갖는 노드들을 상속자 리스트로 포함하고 뿐만 아니라, 그 외의 노드들도 캐쉬로 라우팅 정보를 유지한다. 이 라우팅 정보 중에 캐쉬에 해당되는 노드들의 경우에는 남은 캐쉬 요량이 없을 경우 기존 것들은 LRU(least recently used) 알고리즘에 의해서 새로운 노드 정보로 갱신된다. 그러나 이럴 경우, 추정-지연시간이 짧게 예측된 노드도 제거 될 수 있으므로, 현재의 노드와 새로 추가될 노드 사이의 지연시간과 새로운 노드와 같은 식별자 범위에 있는 노드들 중에 LRU에 의해 선택된 노드와 현재 노드간의 지연시간을 비교해 새로 추가될 노드와 현재 노드 사이의 지연시간이 작거나 같을 경우에 한에 기존의 노드가 제거되게 해야 한다.

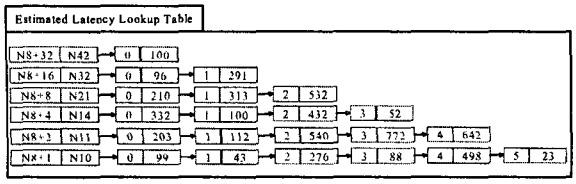

그림 2 추정-지연시간 테이블의 예

3. 성능 평가

3.1 시뮬레이션 환경

본 절에서는 시뮬레이션을 통해 적응적 근접 경로 선택 기법(APRS)의 성능을 CFS의 서버 선택 방법(SS)과 기존의 Chord와의 비교를 통해서 검증하고자 한다. 이를 위해서 링(ring) 토폴로지와 GT-ITM을 사용해 만든 트랜짓-스텁 랜덤 그래프(Transit-Stub Random Graph)[3]를 기반 네트워크로 모델링하여, 노드수의 변화에 따른 쿼리 검색 시간을 측정하였다.

기존의 시뮬레이터에서는 오버레이 상의 모든 노드들 간에 어플리케이션 홉간 지연시간이 일정하게 설정되어 기반 네트워크를 굳이 고려할 필요가 없었다. 그러나 이러한 구조에서는 한 피어에서 다른 모든 피어까지의 네트워크 상의 거리가 같아 홉 카운트가 작을수록 좋은 결과를 얻게 될 수밖에 없다, 그러나 이러한 네트워크 구조는 광대역 네트워크의 노드간 지연시간 특성을 제대로 반영했다고 볼 수 없기 때문에, 본 실험에서는 기존 시뮬레이터를 수정하여 여러 기반 네트워크 토폴로지들 상에서 시뮬레이션 가능하도록 하였다.

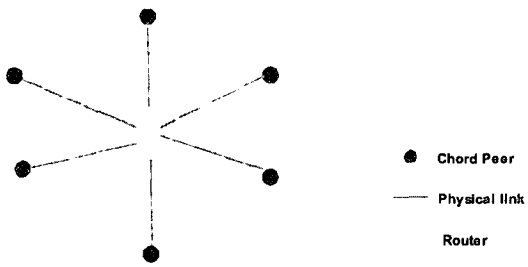

그림 3 기존 Chord 시뮬레이터의 인터넷 토폴로지

표 1 시뮬레이터의 구성

프로그램 명	기 능
Sm	Chord 시뮬레이션 프로그램
traffic_generator	노드의 join/leave 및 데이터의 insertion/deletion을 위한 트래픽 데이터 생성
topology_generator	Ring과 Waxman model을 바탕으로 한 transit-stub 랜덤 그래프 생성
topology_latency	토폴로지 상의 노드간 최단거리 계산

3.2 성능측정

본 실험에서는 시뮬레이션에서 식별자의 비트 수는 20으로 설정하였고, 라우팅 테이블의 크기는 30으로 설정하여, 각 피어들이 알 수 있는 이웃 피어들의 수를 결정하였고, 검색이 완료되기까지 걸리는 지연시간과 검색 라우팅 중 한 홉에 의해 발생하는 지연시간의 크기를 측정하였다. 이때 지연시간은 질의를 생성해서 쿼리 식별자의 상속자를 찾을 때까지의 시간을 의미한다.

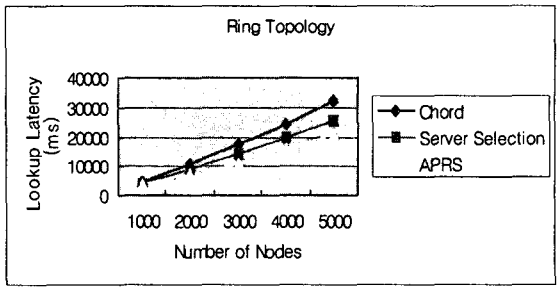

그림 3 네트워크 크기에 따른 검색-지연시간 측정

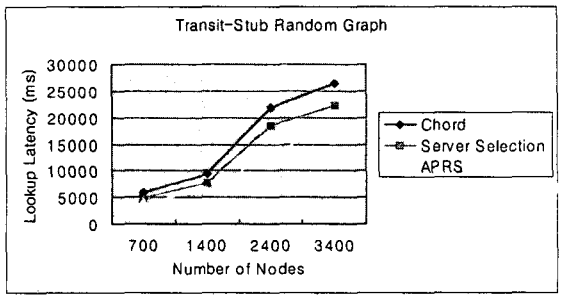

그림 4 네트워크 크기에 따른 홉간-지연시간 측정

4. 결론

본 논문에서는 적응적 근접 라우팅 기법을 이용하여 오버레이 상의 피어들의 기반 네트워크 상의 지연시간을 고려해 라우팅 패스 상의 홉 수가 늘어나더라도, 홉간-지연시간을 향상시켜 전체적인 검색 시간을 줄일 수 있는 피어를 선택한다.

Chord와 이를 개선한 CFS의 서버 선택 기법과 성능을 비교하여 보았다. 그 결과 Chord 검색의 경우 기존 방법에 비해서 링과 랜덤 그래프 상에서는 약 30%의 성능 향상을 가져 왔고, 서버 선택 기법에 대해서는 약 20%의 성능 향상을 보였다. 또한 기반 네트워크에 적응하기 위해서 필요한 시간 역시 충분히 납득할만하다.

참고문헌

[1] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, Hari Balakrishnan, "Chord : A Scalable Peer-to-Peer Lookup Protocol for Internet Applications", *IEEE/ACM Transactions on Networking,* 11(1), pp. 17-32, 20.

[2] Hui Zhang, Ashish Goel and Ramesh Gobindan, "Incrementally Improving Lookup Latency in Distributed Hash Table", *Proceeding of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pp. 114~125, 2003.

[3] *GT-ITM*, http://www.isi.edu/nsnam/ns/ns-topogen.html

[4] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning : An Introduction*, A Bradford Book The MIT, 2002.