DHT 기반 피어-투-피어 시스템을 위한 적응적 근접경로 선택기법

송 지 영[†] · 한 세 영[††] · 박 성 용[†††]

요 약

다양한 네트워크로 구성된 인터넷 환경에서 오버레이 홉 수를 최소화 하는 Chord를 비롯한 기존의 분산 해싱 테이블 기반의 피어-투-피어 시스템에서는 목적 노드까지의 실질적인 질의 라우팅 시간을 줄일 수 없다. 따라서 제안하는 적응적 근접경로 선택기법에서 각 노드는 라우팅 테이블의 엔트리 노드 중 지연시간을 최소로 하는 노드를 선택하여 질의 메시지를 전달하도록 한다. 이를 위하여 각 엔트리의 목적지까지의 지연시간을 큐 라우팅 알고리듬과 지수적 최근-가중치 평균을 이용해 예측하여 검색 테이블로 저장하고, 노드들 간 메시지 전달 시 이 정보를 교환하여 추가 부하 없이 검색 테이블에 반영한다. 시뮬레이션을 통해 제안하는 기법이 전체 검색 시간과 홉 간 지연시간에 있어서 근접성을 고려하지 않은 Chord나 CFS의 서버 선택 알고리듬에 비해 월등히 좋은 성능을 보임을 확인할 수 있다.

키워드 : 피어-투-피어, 분산 해싱 테이블, 근접 경로 선택 방법

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

Jiyoung Song[†] · Saeyoung Han[††] · Sungyong Park[†††]

ABSTRACT

In the Internet of various networks, it is difficult to reduce real routing time by just minimizing their hop count. We propose an adaptive proximity route selection method in DHT-based peer-to-peer systems, in which nodes select the node with smallest lookup latency among their routing table entries as a next routing node. Using Q-Routing algorithm and exponential recency-weighted average, each node estimates the total latency and establishes a lookup table. Moreover, without additional overhead, nodes exchange their lookup tables to update their routing tables. Several simulations measuring the lookup latencies and hop-to-hop latency show that our method outperforms the original Chord method as well as CFS' server selection method.

Key Words : Peer-to-Peer System, Distributed Hashing Table(DHT), Proximity Route Selection(PRS)

1. 서 론

분산 해싱 테이블(Distributed Hashing Table, DHT) 시스템에서 각 노드들은 자신의 라우팅 테이블에서 찾고자 하는 대상의 키 값과 가장 가까운 식별자를 갖는 노드를 다음 라우팅 노드로 선택하는 방법으로 $O(logN)$ (N은 시스템 내의 전체 노드의 수)개의 오버레이 홉 수의 메시지 라우팅으로 해당 노드를 찾아낼 수 있다[1-3]. 그러나 오버레이 네트워크에서 임의의 두 노드 간 라우팅 지연시간은 실제 네트워크에서의 트래픽 지연시간과는 다르다[4]. 예를 들어, 오버레이 네트워크에서 한 홉 떨어진 두 노드는 실제 네트워크에서는 서울과 미국에 위치해 있을 수도 있고, 한 건물 안에 있을 수도 있다. 따라서 기존의 연구들에서 제안하는 오버레이 홉 수를 최소화하는 라우팅 방법으로는 실제적인 라우팅 시간 향상을 기대하기 어렵고, 다음 라우팅 노드를 선택할 때 오버레이 홉 수 뿐 아니라 실제적인 네트워크의 근접성을 고려하는 라우팅 방법이 필요하다.

DHT 시스템에서 실제적인 네트워크의 근접성을 고려하기 위한 기법들은 크게 3가지 기본적인 접근 방법으로 나뉜다[5]. 첫 번째는 근접이웃 선택기법(Proximity Neighbor Selection, PNS)인데, 이는 실제 네트워크에서 가까운 노드들로 각 노드의 라우팅 테이블을 구성하는 방법이다. 이를 위해서 각 노드들은 임의로 주변 노드들을 샘플링 하여 실제 네트워크에서 자신과의 거리를 측정하고, 이를 기반으로

※ 본 연구는 서강대학교 산업기술연구소의 지원으로 수행되었음.
† 정 회 원 : LG전자 단말연구소 연구원
†† 정 회 원 : 서강대학교 컴퓨터학과 박사과정
††† 정 회 원 : 서강대학교 컴퓨터학과 부교수
논문접수 : 2005년 8월 23일, 심사완료 : 2006년 1월 15일

라우팅 테이블을 갱신하는 방법을 취한다. 근접이웃 선택기법은 샘플링 되는 노드의 수가 많아지면, 검색 시간이 거의 실제 네트워크의 지연시간과 거의 같은 정도로 줄어드는 성능 향상의 장점이 있지만, 많은 노드를 샘플링하기 위한 트래픽 증가, 잦은 라우팅 테이블 수정 등 그에 따르는 비용이 크다는 단점이 있다. 최근 연구에 따르면, 근접 이웃 선택을 통한 성능 향상을 위하여, 노드 간 링크의 대역폭이 높거나 이웃 노드의 성능이 좋아서 홉간 지연시간이 가장 적은 이웃 노드를 선택하는 경우, 노드간 연결 간선이 균일하지 않게 되어 노드의 고장이나 보안에 취약하게 된다[6]. 또한 지속적인 프루빙(active probing)을 통해 최적의 이웃노드를 선택해야 하므로, 확장성 있게 노드 간 지연시간을 예측하기 위해서는 프루빙과 네트워크 좌표 알고리듬 사이에 긴밀한 조정이 필요하게 되어 오버헤드가 늘어난다[7].

두 번째는 근접경로 선택기법(Proximity Route Selection, PRS)인데, 이는 이미 구축된 라우팅 테이블의 엔트리에 속한 노드들 중에서, 상대적 근접성을 고려하여 목적지까지 지연시간을 줄이는 최적의 이웃 노드를 선택한다. 일반적으로 근접경로 선택기법은 근접이웃 선택기법에 비해 성능 향상의 폭은 작지만, 라우팅 테이블을 새로 구축할 필요가 없고, 기존의 DHT 방식에 추가적인 비용이 거의 들지 않는다는 장점이 있다.

마지막으로 근접식별자 선택기법(Proximity Identification Selection, PIS)이 있는데, 이는 각 노드에 고유 식별자를 실제 네트워크에서 서로 근접한 노드일수록 더 가깝게 배정하는 방법이다. 그러나 이 방법은 부하를 효율적으로 분산하기 어렵고 쉽게 오류가 발생하는 단점이 있다.

본 논문에서는 두 번째 방법인 근접경로 선택기법의 하나로, DHT기반의 대표적인 P2P 시스템인 Chord를 기반으로 네트워크의 근접성을 고려한 적응적 근접경로 선택기법을 제안하고자 한다. 이 기법은 CFS의 서버 선택기법에서 시스템에 참가하고 있는 전체 노드의 수와 그들 간의 오버레이 네트워크에서의 평균 지연시간을 알아야 하는 한계를 극복하고[8], 전체 오버레이 네트워크의 각 노드들이 자신의 주변 정보만을 이용하여 목적 노드까지의 지연시간을 추정하고, 이를 바탕으로 다음 라우팅 노드를 자신의 라우팅 테이블에서 선택할 수 있도록 한다. 즉, 라우팅 경로를 선택함에 있어서 남은 홉 수를 가장 많이 줄이는 노드와 가장 근접한 노드 간의 트레이드오프(tradeoff)를 고려함으로써 결과적으로 전체 라우팅의 지연시간을 최소화 할 수 있게 한다.

본 논문은 다음과 같이 구성된다. 2장에서는 기존 Chord에서 네트워크의 근접성을 고려한 경로 선택 기법인 CFS의 서버 선택 기법을 소개하고 그 문제점을 나타낸다. 3장에서 제안하고자 하는 적응적 근접경로 선택기법을 소개하고, 4장에서는 기존 Chord와 CFS의 서버 선택 그리고 적응적 근접경로 선택기법을 시뮬레이션하고 이들의 성능을 비교한다. 5장에서는 결론을 도출한다.

2. 관련 연구

CFS는 Chord를 기반으로 하는 읽기 전용 파일을 위한 분산 파일 시스템으로 파일의 블록-레벨 저장소를 제공하는 분산된 노드들로 구성된다[8]. CFS는 세 개의 주된 계층으로 이루어지는데, 최상위에 사용자 혹은 응용 프로그램과의 인터페이스를 위한 파일 시스템 계층이 있고, 그 밑에 DHash라 불리는 분산 블록을 관리하기 위한 계층이 존재한다. 최하위에는 Chord 프로토콜이 구현되어 있는 Chord 계층으로 이루어졌다.

CFS의 클라이언트 파일 시스템은 오버레이 네트워크에 분산되어 있는 블록들을 하나의 파일로 해석하고, 응용 프로그램에게 일반적인 읽기 전용 파일 시스템의 인터페이스를 제공한다. DHash 계층은 클라이언트 파일 시스템에게 데이터 블록을 저장하고 검색할 수 있는 서비스를 제공하며, 블록에 대한 캐시를 저장하는 역할을 한다. 또한 DHash 계층은 블록을 저장할 서버를 선택하거나 클라이언트가 원하는 파일을 구성하는 블록들의 위치를 찾기 위하여 Chord 계층의 서비스를 사용하는데, Chord 계층에서는 해싱 함수를 사용하여 각 서버에게 식별자를 할당하고, 블록의 식별자를 통해 해당 블록의 상속자 서버를 결정한다. Chord 계층은 Chord 프로토콜에서 제공하는 기본 기능 외에도 파일 시스템 성능의 향상을 위해 블록을 찾아내는데 걸리는 시간을 최소화하는 것이 중요하기 때문에, 실제 네트워크에서 거리가 가까운 서버를 선호하여 질의를 라우팅 하는 기법을 사용한다.

이 CFS의 서버 선택 기법은 기본적으로 반복 검색(iterative lookup) 방식으로 구현된다. 블록을 찾고자 하는 노드를 n이라고 할 때, n은 자신이 알고 있는 이웃 노드 j에게 RPC 메시지를 보내서 j의 라우팅 테이블과 j와 라우팅 테이블 엔트리의 노드 사이에 측정된 지연시간 정보를 가져온다. 이를 위해 노드 j는 자신의 라우팅 테이블을 만들 때 지연시간을 미리 측정해 그 정보를 저장한다. 노드 n은 이 정보를 바탕으로 다음 질의 메시지를 보낼 노드n_i에 대한 지연시간을 다음의 <식 1>에 의해서 계산하여 예측한다.

$$C(n_i) = d_i + \bar{d} \times H(n_i) \qquad <식\ 1>$$

d_i는 현재의 노드 n과 n_i사이의 지연시간을 의미하고, $\bar{d}$는 노드 n이 예측하는 전체 시스템에서의 노드 간 지연시간에 대한 평균값이다. $H(n_i)$는 n_i이후에 남은 홉 수에 대한 추정 값을 나타내고, 이것은 다음의 <식 2>에 의해서 구해진다.

$$H(n_i) = \log\left(\frac{n_i - k}{2^{160}} \times N\right) \qquad <식\ 2>$$

여기서 N 은 노드 n이 추정한 오버레이 네트워크의 전체 노드의 수, 160은 CFS 시스템에서 사용하는 식별자의 비트

길이를 나타낸다. 이때 노드 n은 N을 추정하기 위해서 이웃 노드들의 식별자의 밀도를 고려한다. 이렇게 예측된 N에 찾고자하는 블록의 키 k와 질의를 보내고자 하는 노드 n_i사이의 거리가 전체 공간에서 차지하는 비율을 곱하여 n_i와 목표 노드 사이에 있는 노드들의 수를 구한다. 여기에 2를 밑으로 하는 로그를 취함으로써 질의를 목표 노드로 전달하는데 필요한 라우팅 홉 수를 예측할 수 있다. 이것은 Chord 검색 알고리즘에 의해서, 한 홉 당 절반 정도의 노드를 줄일 수 있기 때문이다. 이렇게 예측된 홉의 수에 오버레이 노드들 간의 평균 지연시간을 곱하고 여기에 j와 n_i 사이의 지연시간 d_i을 더함으로써, 결과적으로 n_i를 선택할 때 목표 노드까지 검색이 진행되는데 걸리는 전체 검색 시간을 추정할 수 있다. CFS의 서버 선택 메커니즘은 질의를 라우팅할 다음 노드로 이 추정된 검색 시간 $C(n_i)$를 최소화하는 노드를 선택한다.

CFS에서 서버 선택을 위한 정확한 지연시간을 예측하기 위해서는 시스템에 참가하고 있는 전체 노드의 수와 그들 간의 오버레이 네트워크에서의 평균 지연시간을 필요로 하고, 이것을 얼마나 정확하게 추정할 수 있는지에 따라서 서버 선택 방법의 성능이 결정된다. CFS에서는 이 값을 현재 자신이 알고 있는 노드들로부터 얻은 정보를 토대로 추정하는데, 네트워크의 크기가 커질수록 한 노드가 알고 있는 다른 노드들에 대한 정보는 대표성이 떨어지게 되므로, 결과적으로 정확한 값에 대한 추정이 어려워지게 된다. 또한, 네트워크 규모가 클수록 노드들 간의 지연시간에 대한 편차 역시 커지게 된다.

결론적으로 CFS의 서버 선택 방법은 광대역 네트워크에서 검색 지연시간을 향상시켜주는 방법이라기보다는 중소 규모의 네트워크에서 데이터 블록에 대한 검색 시간을 줄이고, 블록을 가장 지연시간이 적은 노드로부터 전송 받기 위한 방법이라 생각된다. 따라서 광대역 네트워크에서 실제 데이터에 대한 메타 데이터를 찾아야 하는 전형적인 P2P 시스템에서는 질의 메시지 전달 시간을 향상시키기 힘들 것으로 예상된다. 따라서 본 논문에서는 지수적 최근-가중치 평균을 이용하는 적응적 근접경로 선택기법을 제안하고자 한다.

3. 적응적 근접경로 선택기법

3.1 지수적 최근-가중치 평균

지수적 최근-가중치 평균의 특징은 현재의 값에 가장 큰 비중을 두면서도 동시에 과거 측정값에 대해서도 지수적인 가중치를 두어 어느 정도 고려하기 때문에, 현재 측정된 값이 급격히 증가하거나 감소할 경우에도 이에 따라 예측 값이 급격히 변하지 않는다는 점이다. 이는 동적으로 변하는 링크의 특성을 반영하면서도 동시에 오버레이 홉이 지니고 있는 기본적인 링크 특성 역시 반영할 수 있게 해준다. 다음의 (그림 1)은 본 기법에서 사용하는 지수적 최근-가중치 평균의 일반적인 식을 나타낸다. 여기서 $Q_{n,k}$는 노드 n에서 k번째 추정 값을 의미한다.

$$
\begin{aligned}
Q_{n,k} &= \alpha \cdot cost_k + (1-\alpha) \cdot Q_{n,k-1} \\
&= \alpha \cdot cost_k + (1-\alpha) \cdot \alpha \cdot cost_{k-1} \\
&\quad + (1-\alpha)^2 \cdot Q_{n,k-2} \\
&= \cdots \\
&= \alpha \cdot cost_k + (1-\alpha) \cdot \alpha \cdot cost_{k-1} \\
&\quad + (1-\alpha)^3 \cdot \alpha \cdot cost_{k-3} + \cdots \\
&\quad + (1-\alpha)^{k-1} \cdot \alpha \cdot cost_1 + (1-\alpha)^k \cdot Q_{n,0} \\
&= (1-\alpha)^k \cdot Q_{n,0} + \sum_{i=1}^{k} \alpha \cdot (1-\alpha)^{k-i} \cdot cost_i
\end{aligned}
$$

(그림 1) 지수적 최근-가중치 평균의 식

여기서, 마지막 식의 계수의 합이 $(1-\alpha)^k + \sum_{i=1}^{k} \alpha \cdot (1-\alpha)^{k-i} = 1$ 이 되어 계수가 가중치의 역할을 함을 알 수 있다. 즉, $\alpha \cdot (1-\alpha)^{k-i}$ 에 의해서 현재 k번째 단계에서 $cost$ 가 측정된 시점 i 까지 얼마나 떨어져 있는지 $k-i$ 에 따라서 가중치가 결정된다. α가 $0 < \alpha \leq 1$ 사이의 값이라고 할 때 1 - α는 1보다 작으므로 i 시점에서 측정된 $cost$ 값의 비중은 k 가 증가함에 따라 지수적으로 감소하게 된다.

3.2 지연 시간 추정을 통한 라우팅 피어 선택법

제안하는 적응적 근접경로 선택기법은 재귀적 검색(Recursive Lookup) 방식으로 동작하며, 각 노드는 라우팅 테이블의 노드 중 지연시간을 최소로 하는 노드를 선택하여 질의 메시지를 전달하고, 질의가 목적지에 도달할 때 까지 계속 된다.

노드 n이 질의 q 를 받는다고 하자. 이때 q가 m-비트의 식별자 공간에서 $[n+2^i, n+2^{i+1})$에 속한다고 하면 하면 $(0 \leq i \leq m-1)$, 이를 질의 q가 i번째 상태에 있다고 정의한다. 이때, 노드 n이 질의 q를 전달 할 다음 라우팅 노드가 될 후보 노드들의 집합 $c\_set_n(q)$ 는 다음 <식 3>과 같다.

$$
\begin{aligned}
c\_set_n(q) = \{&\text{node x} \mid \forall x \text{ in the routing table} \\
&\text{which satisfies its ID} \in (n,q]\}
\end{aligned}
\qquad \text{<식 3>}
$$

노드 n은 이 후보 노드들 중에서 목적지까지의 추정-지연시간을 최소화 하는 노드를 질의를 라우팅 할 다음 노드로 선택한다. 추정-지연시간을 최소화하는 노드를 선택하기 위해서 적응적 근접경로 선택기법은 라우팅 테이블에서 식별자 x를 가진 각 엔트리에 대하여 다음과 같은 추가적인 정보를 필요로 한다.

$$
(x, Q_n(x, i), Q_n(x, i+1), \ldots, Q_n(x, m))
$$

여기서 $Q_n(x,i)$는 i번째 상태인 질의에 대해 다음 노드로 x를 선택했을 경우 추정-지연시간을 의미한다. 라우팅 테이블의 각 엔트리에 대해 log(N)개의 추가적인 정보가 필요하

고, 라우팅 테이블은 log(N)개의 엔트리를 포함하므로, 전체적으로 $\log^2(N)$의 저장 공간이 추가적으로 필요하게 된다. 이 추가적인 정보가 포함된 테이블을 검색 테이블(lookup table)이라고 하고, 변경되는 엔트리의 내용은 다음 <식 4>와 같다.

$$Q_n^{new}(x,i) = (1-\alpha) \cdot Q_n^{old}(x,i) + \alpha \cdot (\text{observed latency to } x + \min_{z \in \text{neighbor of } x} Q_x(z,i)) \quad <식\ 4>$$

위의 식은 분산 패킷 라우팅 알고리즘에서 강화 학습을 기반으로 한 큐 라우팅(Q-Routing) 알고리즘 [9]에서 기인하고, α는 $0 < \alpha < 1$ 의 값을 갖는다.

이웃 노드로부터 받은 정보를 통해 검색 테이블을 변경하는 방법에는 두 가지가 있다. 원래 Chord 프로토콜에서는 시스템 상에 있는 각 노드가 라우팅 테이블의 정보를 갱신하기 위해 안정화 프로토콜을 정기적으로 실행한다. 이 때 $Q_x(z, i)$ 에 해당하는 추가 정보를 모으고, 관찰된 지연 시간들에 의해 x 에 대한 지연 시간을 측정 할 수 있다. 이 기법은 제어를 위한 추가적인 메시지를 필요로 하지 않는다는 장점을 갖는다. 또 다른 하나의 변경 방법은 선택된 이웃 노드에게 질의 메시지를 전달하고, 그 이웃 노드로부터 답변을 받아 새롭게 추정된 지연시간을 적용하여 검색 테이블을 갱신하는 것이다. 질의를 전달 할 다음 노드를 선택하는 방법은 그리디(Greedy)한 방법을 사용하는데, 만일 노드 n이 i번째 상태라고하면, 그리디 선택 방법은 다음 <식 5>와 같다.

$$\pi_n(i_q) = \arg\min_{c \in candidate_n(q)} Q_n(c, i_q) \quad <식\ 5>$$

이 때 $\pi_n(i_q)$는 라우팅 작업이 진행 되는 동안의 노드 n에 대한 라우팅 정책이 된다.

3.3 라우팅 테이블 관리

피어-투-피어 시스템은 동적인 시스템으로 기존 노드가 시스템에서 나가거나, 새로운 노드가 추가 될 수 있는데. 각 노드의 라우팅 테이블의 그 크기가 제한되어 있으므로 한 노드에서 유지할 수 있는 이웃 노드의 수는 제한되어 있다.

실제 Chord의 구현의 경우 확장성 있는 검색을 위하여 기본적으로 핑거 테이블 엔트리의 노드들 뿐 아니라, 식별자 공간에서 자신의 식별자 바로 다음으로 큰 식별자를 갖는 노드들을 상속자 리스트로 구축하여 관리하고, 그 외의 노드들도 캐시로 라우팅 정보를 유지한다. 이 중 캐시의 경우 남은 메모리의 용량이 없을 경우 LRU(Least Recently Used) 알고리즘에 의해서 기존 정보를 새로운 노드 정보로 갱신시킨다. 그러나 이 때 기존 정보 중 추정-지연시간이 짧은 노드에 대한 정보가 삭제 될 수 있으므로, 현재 노드와 새로 추가될 노드 사이의 지연시간과 LRU에 의해 삭제

```
//노드 n은 식별자 r -> x에 대한 검색 요청 r을 처리한다.
processRequest (Node n, Request r)
    //만일 n이 r->x의 상속자 바로 앞 선임자라면
    if r->x exists between (n -> id, successor (n))
        //요청 r을 생성한 노드에 successor(n)의 라우팅 정
          보를 전달한다.
        sendSuccessor (n, r -> initiator, successor)
    //아니면
    else
        //식별자 r -> x의 범위 i를 구한다.
        i = getIDRange (n, r -> x)
        //i안에서 가장 작은 추정 지연 시간을 갖는 이웃피
          어 min_n을 찾는다.
        min_n = findMinEstimatedLatencyNeighbor
                (n -> fingerList, i)
        //min_n에게 요청 r을 전달하고 새로운 추정
          지연시간을 계산한다.
        new_estimated_latency = sendRequest
                                (n, min_n, r)
        //새로운 추정 지연 시간으로 업데이트 규칙에 따라
          테이블을 갱신한다.
        updateEstimatedLatencyTable
        (n -> fingerList, i, new_estimated_latency)
```

(그림 2) 근접경로 선택기법의 의사 코드

되도록 선택된 노드와 현재 노드 사이의 지연시간을 비교해 새로 추가될 노드와 현재 노드 사이의 지연시간이 작거나 같을 경우에만 기존의 노드가 제거되도록 해야 한다. (그림 2)에서 근접경로 선택기법의 의사코드를 나타내었다.

5. 시뮬레이션 성능 평가

본 장에서는 시뮬레이션을 통해 적응적 근접경로 선택기법의 성능을 CFS의 서버 선택 기법과 기존의 Chord와의 비교를 통해서 검증하고자 한다. 이를 위해서, 링 토폴로지와 GT-ITM을 사용해 만든 트랜짓-스텁 랜덤 그래프(Transit-Stub Random Graph)[10]를 기반 네트워크로 모델링하여, 노드 수의 변화에 따른 질의 검색 시간을 측정하였다.

5.1 시뮬레이션 환경

본 실험에서는 Chord의 SFS-시뮬레이터[11]를 수정하여 이를 바탕으로 실험을 진행하였다. 시뮬레이터는 이벤트-구동 방식으로 동작하며 시스템 내에서 데이터 아이템의 삽입과 삭제, 새로운 노드가 추가되거나 기존 노드가 떠나는 것을 시뮬레이션 할 수 있다. 뿐만 아니라 안정화 프로토콜과 검색 메시지의 재전송 메커니즘 등도 포함되어 있어 Chord의 모든 기본 프로토콜들이 시뮬레이션 시에 고려될 수 있다.

기존의 시뮬레이터에서는 오버레이 상의 모든 노드들 간에 오버레이 홉 간 지연시간이 일정하게 설정되어 기반 네트워크를 고려할 수 없었고, 따라서 한 노드에서 다른 모든 노드까지의 거리가 모두 같아 홉 수를 최대한 줄이는 노드의 선택이 최상의 결과를 얻게 될 수밖에 없다. 그러나 이러한 네트워크 구조는 광대역 네트워크의 노드 간 지연시간

특성을 제대로 반영했다고 볼 수 없으므로, 본 실험에서는 기존 시뮬레이터를 수정하여 여러 기반 네트워크 토폴로지들 상에서 시뮬레이션이 가능하도록 하였다.

5.1.1 네트워크 토폴로지

시뮬레이션의 현실성을 결정하는 가장 중요한 요소 중의 하나는 시뮬레이션에서 사용되는 인터넷 토폴로지이다. 본 실험에서는 인터넷 토폴로지에 대한 최근 연구들을 바탕으로 다음과 같은 두 가지 토폴로지를 사용하여 본 논문에서 제시한 피어 선택 기법의 성능을 측정하였다.

첫 번째로 고려한 토폴로지는 링 형 토폴로지이다. 최근의 연구에 의하면, 링 형 토폴로지의 지연시간 증가가 인터넷 상 노드들의 지리적 거리와 관계된 지연시간 증가 특성과 가장 잘 맞는다고 알려져 있다 [4, 12]. 본 실험에서 사용한 링 토폴로지에서 각 노드들은 모두 네트워크에서 일정한 거리가 떨어져 있다고 가정하고, 각 노드 간의 지연시간은 이 거리를 바탕으로 최소 경로로 구해진다.

두 번째로 고려한 토폴로지는 GT-ITM 인터넷 토폴로지 생성기에서 만드는 트랜짓-스텁 형의 랜덤 그래프(Transit-Stub Random Graph)이다. 이 랜덤 그래프는 트랜짓 도메인들과 스텁 도메인들로 이루어진 계층적인 형태를 갖는다.

이 때 그래프 상의 각 노드 사이의 간선의 수는 다음의 확률에 의해서 결정되는데, 이것은 Waxman[13]에 의해 최초로 제안된 모델을 수정한 것으로 일반적으로 많이 사용되는 랜덤 그래프 모델 중에 하나이다 (<식 6> 참조).

$$P(u,v) = \alpha \cdot e^{-d/(L-d)} \qquad <식\ 6>$$

여기서 α는 $0 < \alpha \leq 1$ 사이의 상수이고, d는 노드 u와 v 사이의 유클리드 거리를 의미한다. 그리고 L은 $L = \sqrt{2} \times scale$로 그래프 상의 임의의 두 노드간의 최대 거리를 나타낸다. 따라서 α가 커질수록 그래프 상의 간선의 수는 증가하게 되고, L이 커질수록 노드간의 지연시간은 상대적으로 크게 측정되게 된다. 본 실험에서는 α를 1로 설정하였고, L은 GT-ITM의 설정값으로 실험하였다.

5.1.2 시뮬레이션 방법

본 실험에서 시뮬레이터는 앞서 언급된 링 형과 랜덤 그래프 네트워크 토폴로지를 시뮬레이터의 입력으로 받아 Chord 오버레이 네트워크의 기반 네트워크의 역할을 하도록 한다. 이 때 시뮬레이터는 Chord 오버레이의 각 노드를 입력 받은 그래프상의 노드로 랜덤하게 매칭시킨다.

이와 같은 과정을 거친 후에 시뮬레이터는 미리 생성된 트래픽 정보에 따라서 다음과 같이 시뮬레이션을 수행하게 된다. 먼저, 1초에 하나씩 N개의 노드들을 시스템으로 참여시킨다. 이 때 각 노드들은 다른 노드에 대한 기본적인 라우팅 테이블을 형성하게 된다. 이 과정이 끝나면 각 노드에게 동일한 확률을 가지고 노드 당 평균 50개의 데이터를 삽입시켜, 각 노드가 평균 50개의 데이터에 대한 상속자가 되게 한다. 이제 각 노드는 동일한 확률로 평균 50개의 데이터를 선택하여 이들에 대한 검색 요청을 수행한다. 이 과정에서부터 노드 검색 기법이 적용되어, 각 아이템의 식별자 범위에 따른 추정-지연시간을 측정하고, 이로부터 라우팅 될 노드를 선택하면서 시뮬레이션이 진행되게 된다. 이때 요청은 시뮬레이션 시간으로 1초마다 발생하게 된다. 마지막으로 모든 질의 요청에 대한 처리가 완료되면 시뮬레이션은 끝나게 된다.

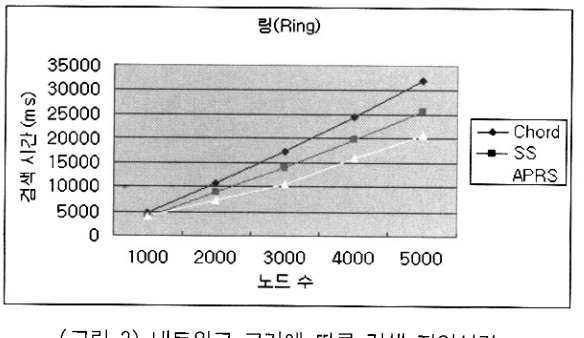
5.2 시뮬레이션 결과

본 실험에서는 Chord 검색이 완료되기까지 걸리는 지연시간과 질의 라우팅 중 한 홉에서 발생하는 지연시간의 크기를 측정하였다. 이때 지연시간은 질의를 생성해서 질의 식별자의 상속자를 찾을 때까지 걸리는 시간을 의미한다.

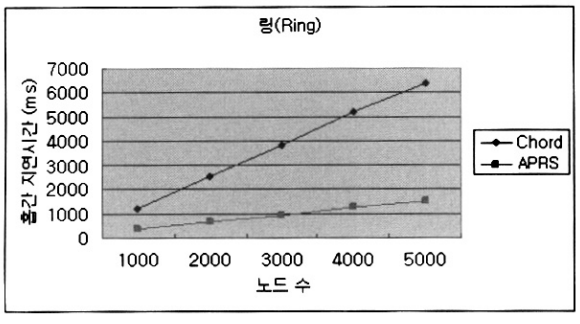
시뮬레이션에서 식별자의 비트 수는 20으로 설정하였고, 라우팅 테이블의 크기는 30으로 설정하여, 각 노드가 알 수 있는 이웃 노드들의 수를 결정하였다. 본 실험에서 적응적 근접경로 선택기법과 기존 Chord와의 비교를 통하여 네트워크의 근접성을 고려한 라우팅의 효과를 검증하고, 기존의 근접 라우팅 기법인 CFS의 서버 선택 기법과의 비교를 통하여 성능의 향상을 확인하고자 하였다. 또한 기존 Chord에 대해서 검증된 분석적 결과와의 비교를 통해서 본 실험의 시뮬레이션 과정이 올바로 이루어졌는지 입증하고자 한다.

5.2.1 링 토폴로지

먼저 링을 기반 네트워크로 성능을 측정한 실험 결과를 나타내고자 한다. 링은 앞서 언급되었듯이 인터넷의 지연시간 증가 특성과 가장 유사한 특징을 지닌 그래프이다. (그림 3)에서는 링 토폴로지 상에서 노드의 크기를 변화시켜감에 따라서 적응적 근접경로 선택기법(APRS), CFS의 서버 선택 기법(SS), 기존 Chord에서의 질의 검색 시간을 나타내었다. 모든 노드에 일정하게 데이터 아이템이 분산되어 있고, 동일한 확률로 랜덤하게 선택된 식별자에 대해서 질의를 요청할 경우 Chord는 네트워크의 크기 N에 대해서 *(평균 홉 간-지연시간) X O(log N) / 2* 로 검색 지연시간의 분석적 예측이 가능하다[1]. 이 분석적 방법에 의해 계산된 결과는 (그림 3)의 실제 실험 결과와 1.6%~2.2% 범위 내에서 거의 일치하였고, 이를 통해서 시뮬레이션의 유효성을 확인할 수 있다.



(그림 3) 네트워크 크기에 따른 검색-지연시간



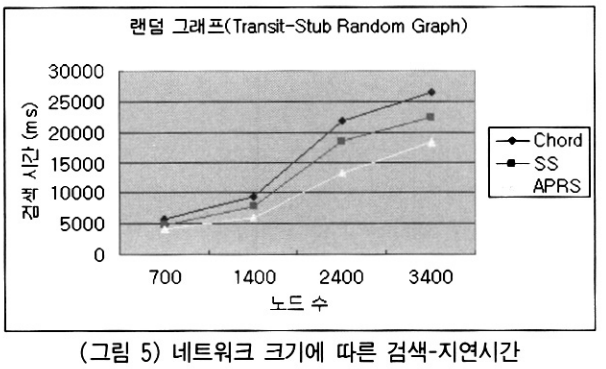
(그림 4) 네트워크 크기에 따른 홉간-지연시간

(그림 4)에는 Chord의 전체 검색 과정 중에 한 홉에서의 평균 지연시간을 나타낸다. 적응적 근접경로 선택기법의 경우 검색 홉 수가 늘어나더라도 가까운 이웃 노드를 통해서 질의를 전달하여 전체적인 검색 시간의 향상을 추구하므로, 예상대로 근접성을 고려하지 않은 기존 Chord의 기법에 비해 상대적으로 홉 간-지연시간이 작음을 알 수 있다.

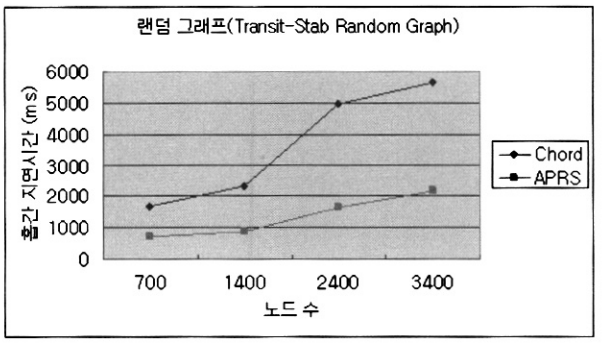
(그림 3)과 (그림 4)의 결과를 종합해보면, 적응적 근접경로 선택기법은 의도했던 바와 같이 홉간-지연시간의 향상을 통해서 전체적인 검색 작업의 검색-지연시간을 줄인다는 것을 알 수 있었다. 실험 결과 적응적 근접경로 선택기법은 기존 Chord에 비해 네트워크의 크기에 따라서 약 30%의 성능 향상을 보였고, CFS 서버 선택 기법에 비해서는 약 20%의 성능 향상이 있음을 확인할 수 있다. 또한, 네트워크의 크기가 작은 경우보다 네트워크 크기가 커질수록 더 높은 성능 향상을 보이는 경향을 확인할 수 있다. 이것은 네트워크 크기가 작을 경우 홉간-지연시간의 차이가 작아 근접 이웃 노드를 선택함으로써 얻을 수 있는 성능 개선의 여지가 상대적으로 줄어들기 때문이다.

5.2.2 랜덤 그래프

이번 절에서는 트랜짓-스텁 구조의 랜덤 그래프를 기반 네트워크로 모델링하여 성능을 비교한 결과를 나타내었다. 전체적인 결과는 링을 기반 네트워크 모델로 사용했을 때와 비슷한 경향을 보인다. (그림 5)에서 알 수 있듯이 랜덤 그래프의 경우에도 네트워크의 크기가 커질수록 성능 향상의 폭이 커지는 경향을 보였다. 이것은 제안하는 적응적 근접경로 선택기법(APRS)이 네트워크의 크기가 증가하더라도 홉 간-지연시간을 줄일 수 있는 근접한 이웃 노드로 검색 메시지를 라우팅 함으로써 전체적인 검색 시간을 향상시켰기 때문이다. (그림 6)에서 랜덤 그래프에서의 네트워크 크기에 따른 홉 간 지연시간의 분포를 나타내었다. 적응적 근접경로 선택기법은 기존 Chord에 비해서 네트워크 크기가 커지더라도 라우팅 시 홉간 지연시간이 크게 증가하지 않음을 알 수 있다. 랜덤 그래프의 경우 적응적 근접경로 선택기법은 기존 Chord와 비교해서 네트워크의 크기에 따라 33%의 성능을 향상시켰고, CFS의 서버 선택 기법(SS)에 비해서는 약 22%정도의 성능 향상을 확인할 수 있다. 링 토폴로지에 비해서 랜덤 그래프에서의 성능 향상이 소폭 더 컸



(그림 5) 네트워크 크기에 따른 검색-지연시간



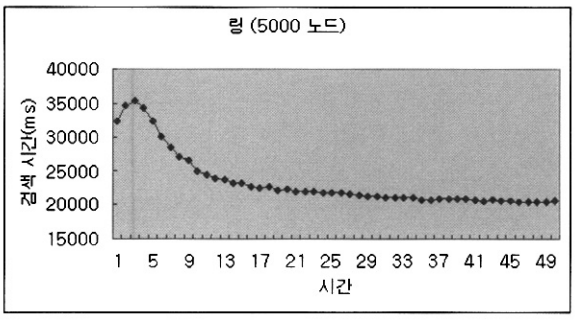
(그림 6) 네트워크 크기에 따른 홉 간-지연시간

는데, 그 이유는 랜덤 그래프의 트랜짓-스텁 구조 때문에 두 노드간의 지름길이 존재할 수 있어 지연시간을 더 단축시킬 수 있기 때문이다.

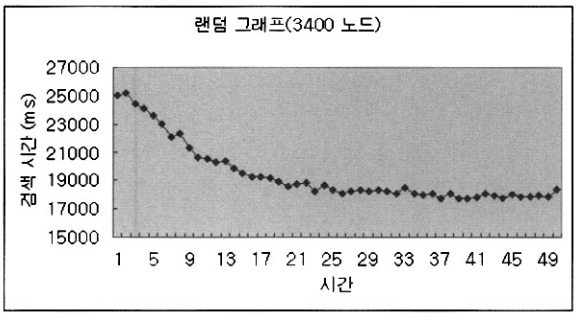
5.2.3 수렴 시간

본 절에서는 제안하는 적응적 근접경로 선택기법에서 시뮬레이션 동안 얼마나 빨리 기반 네트워크에 적응하여 안정상태로 수렴할 수 있는지를 나타내고자 한다.

적응적 근접경로 선택기법은 자신의 이웃 노드들로부터 얻은 주변 정보를 통해 데이터 아이템의 상속자를 찾는데 걸리는 전체 검색시간을 추정하기 때문에 정보가 충분치 않은 시뮬레이션 초기에는 본래의 성능을 내지 못하고, 충분한 정보를 얻을 때까지 어느 정도의 시간이 필요하다. (그림 7)과 (그림 8)에서 각각 5000개의 노드로 이루어진 링 토폴로지와 3400개의 노드로 이루어진 랜덤 그래프에서 시간의 흐름에 따른 질의 요청에 대한 검색 시간의 변화를 나타내었다. 여기서 한 단위의 시간은 시스템 내의 모든 노드들이 1개의 검색 요청을 완료한 시점을 의미한다. (그림 7)에 따르면, 5000개의 노드를 갖는 링 토폴로지의 경우 약 21개의 검색 요청이 처리된 이후에 검색 시간이 안정화됨을 알 수 있고, (그림 8)에서는 3400개의 노드를 갖는 랜덤 그래프의 경우 평균 22개의 질의 요청 이후부터 검색 시간이 수렴하여 안정화되는 것을 확인할 수 있다. 실제 네트워크에서의 실험과 시뮬레이션과는 차이가 있을 수 있으나, 이 정도의 수치는 검색에 대한 요청이 빈번하게 일어나는 피어-투-피어 시스템의 경우 충분히 납득할 만 한 수치라고 할 수 있다.



(그림 7) 링 토폴로지에서 수렴 시간



(그림 8) 랜덤 그래프에서 수렴 시간

6. 결 론

기존 Chord 검색 방법에서는 단순히 오버레이 상의 홉 수를 줄이는 노드를 선택하는데 비해, 기반 네트워크에서 지연시간을 고려해 라우팅 경로의 홉 수가 증가하더라도 홉간-지연시간을 줄여서 전체적인 검색 시간의 줄일 수 있는 노드를 선택하는 방법이 필요한데, 그 대표적인 방법이 CFS의 서버 선택기법이다. 그러나 이 방법에서는 시스템 전체의 노드 수와 각 홉간 평균 지연시간을 알아야 하므로, 각 노드들이 자신의 주변 정보만을 이용하여 목적 노드까지의 지연시간을 추정하고, 이를 바탕으로 다음 라우팅 노드를 자신의 라우팅 테이블에서 선택할 수 있는 확장성 있는 적응적 근접경로 선택기법을 제안하였다.

제안하는 적응적 근접경로 선택기법에서는 각 노드들이 식별자에 따른 추정-지연시간을 지수적 최근-가중치 평균을 이용해 예측하고, 이 정보들을 기반으로 질의 메시지를 받았을 때, 이를 전달할 다음 라우팅 노드를 결정한다. 이를 위해서 $\log^2(N)$의 추가적인 저장 공간을 필요로 하지만, 기존의 Chord는 물론이고 CFS의 서버 선택 기법에 비해 월등한 검색 지연시간과 홉간 지연시간을 가짐을 시뮬레이션을 통해 확인할 수 있었다. 즉, Chord 검색과 비교해서 링과 랜덤 그래프에서 약 30%의 성능 향상을 보였고, 서버 선택 기법에 비해서는 약 20%의 성능 향상을 보였다. 또한 기반 네트워크에 적응하기 위한 시간 역시 충분히 납득할 만한 시간이었다.

그러나 제안하는 적응적 근접경로 선택기법은 Chord 프로토콜에 의해서 이미 구축된 라우팅 테이블을 기반으로 데이터의 식별자에 대한 최적 노드를 선택하므로, 최악의 경우 자신과 먼 네트워크에 속한 노드들에 대한 라우팅 정보만을 라우팅 테이블에 가지고 있어, 결과적으로 홉 간 지연시간을 줄일 수 없게 될 수도 있다. 따라서 보다 나은 성능 향상을 위해서 각 노드의 라우팅 테이블을 효과적으로 구축하고 관리하는 보완에 대한 연구가 앞으로 더 필요하다.

참 고 문 헌

[1] I. Stoica, M. Robert, L. N. David, R. David, M. Karger, K. Frans, D. Frank, and B. Hari, "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications," IEEE/ACM Transactions on Networking, Vol.11, Issue 1, pp.17-32, 2003.

[2] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," in Proceeding of the 18th IFIP/ACM International Conference on Distributed Systems Platforms, 2001.

[3] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing," Technical Report CSD-01-1141. 2001.

[4] H. Zhang, A. Goel, and R. Gobindan, "Incrementally Improving Lookup Latency in Distributed Hash Table," in Proceeding of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pp.114-125, 2003.

[5] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, "The Impact of DHT Routing Geometry on Resilience and Proximity," in Proceeding of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, pp.381-394, 2003.

[6] B. G. CHun, B.Y. Zhao, and J. D. Kubiatowicz, "Impact of Neighbor Selection on Performance and Resilience of Structured P2P Networks," In Proceedings of the forth IPTPS, 2005.

[7] K. P. Shanahan and M. J. Freedman, "Locality Prediction for Oblivious Clients," In Proceedings of the forth IPTPS, 2005.

[8] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-Area Cooperative Storage with CFS," in Proceeding of the 18th ACM Symposium on Operating Systems Principles, pp.202-215, 2001.

[9] J. Boyan and M. Littman, "Packet Rrouting in Dynamically Changing Networks: A Reinforcement Learning Approach," in Proceedings of Neural Information Processing Systems, pp.982-988, 1994.

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

[11] Chord Simulator, http://pdos.lcs.mit.edu/cgi-bin/cvsweb.cgi/sfsnet/simulator

[12] N. P. Venkata and S. Lakshminarayanan, "An Investigation of Geographic Mapping Techniques for Internet Hosts," in Proceeding of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, pp.173-185, 2001.

[13] B. M. Waxman, "Routing of Multipoint Connections," IEEE Journal on Selected Areas in Communications, No.9, pp.1617-1622, 1988.

송 지 영

e-mail : itsuki@lge.com
2002년 서강대학교 물리학과(이학사)
2004년 서강대학교 컴퓨터학과 (공학석사)
2004년~현재 LG전자 단말연구소 연구원
관심분야 : 피어투피어 컴퓨팅, 분산처리 시스템

한 세 영

e-mail : syhan@sogang.ac.kr
1991년 포항공과대학교 수학과(이학사)
2003년 서강대학교 정보통신대학원 (공학석사)
2004년~현재 서강대학교 컴퓨터학과 박사과정 재학중
1996년~현재 (주)이엔지 기술본부
관심분야 : 피어투피어 컴퓨팅, 분산처리 시스템

박 성 용

e-mail : parksy@sogang.ac.kr
1987년 서강대학교 컴퓨터학과(공학사)
1994년 미국 Syracuse University 대학원 (공학석사)
1998년 미국 Syracuse University (공학박사)
1998년~1999년 미국 Bell Communication Research 연구원
1999년~현재 서강대학교 컴퓨터학과 부교수
관심분야 : Autonomic Computing, Peer to Peer Computing, High Performance Cluster Computing and System