DOI: 10.3745/KIPSTC.2009.16-C.1.73

트래픽 및 요청 지연을 최소화한 향상된 리액티브 Chord

윤 영 효[†] · 곽 후 근[††] · 김 정 길[†††] · 정 규 식[††††]

요 약

Chord 방식의 피어들은 라우팅 테이블을 최신으로 유지하기 위해 주기적으로 메시지를 보낸다. 모바일 P2P 네트워크에서 Chord 방식의 피어들은 라우팅 테이블을 최신으로 유지하고 요청 실패를 줄이기 위해서 빠른 주기로 메시지를 보내야 한다. 하지만 이로 인해, 전체 네트워크의 트래픽은 증가하게 된다. 본 연구자들은 기존 연구에서 리액티브 라우팅 테이블 업데이트 방식을 이용하여 기존 Chord에서의 라우팅 테이블 업데이트에 따른 부하를 줄이는 기법을 제안하였으나, 초당 요청 메시지 개수가 많아지게 되면 기존의 방식보다 트래픽 양이 많아지게 되는 단점과 요청처리시간 지연이 기존방식보다 떨어진다는 단점을 가진다.

이에 본 논문에서는 요청처리 시간을 줄이기 위한 향상된 리액티브 라우팅 테이블 업데이트 방식을 제안한다. 기존에 제안된 방식은 요청이 들어올 때마다 테이블을 업데이트 하는 반면, 제안된 방식은 요청이 들어 왔을 때 현재의 테이블 정보가 최신정보인지를 확인하고 만일 아니라면 테이블을 업데이트하는 방식이다. 실험은 버클리 대학에서 만들어진 Chord 시뮬레이터(I3)를 이용하여 수행하였고, 실험을 통하여 제안된 방식이 기존 방식에 비해 성능이 향상되었음을 확인하였다.

키워드 : Chord, 모바일 P2P 네트워크, 리액티브 라우팅 데이블 업데이트

An Improved Reactive Chord for Minimizing Network Traffic and Request Latency

Younghyo Yoon[†] · Hukeun Kwak[††] · Cheongghil Kim[†††] · Kyusik Chung[††††]

ABSTRACT

The peers in the Chord method send messages periodically to keep the routing table updated. In a mobile P2P network, the peers in the Chord method should send messages more frequently to keep the routing table updated and reduce the failure of a request. However this results in increasing the overall network traffic. In our previous method, we proposed a method to reduce the update load of the routing table in the existing Chord by updating it in a reactive way, but there were disadvantages to generate more traffic if the number of requests per second increases and to have more delay in the request processing time than the existing Chord.

In this paper, we propose an improved method of reactive routing table update to reduce the request processing time. In the proposed method, when a data request comes, the routing table is updated only if its information is not recent while it is always updated in the previous method. We perform experiments using Chord simulator (I3) made by UC Berkely. The experimental results show the performance improvement of the proposed method compared to the existing method.

Keywords : Chord, Mobile P2P Network, Reactive Routing Table Update

1. 서 론

Peer-to-Peer(P2P)는 기존의 인터넷에서 사용되던 클라이언트-서버 환경의 단방향 특성을 극복하고, 서버의 문제로 인해 사용자들이 서비스를 받지 못하게 되는 단일장애점(Single Point of Failure) 문제를 해결하기 위해 고안된 분산 컴퓨팅 모델에 기반한 네트워킹 기술이다. P2P 어플리케이션은 정보공유, 파일공유, 대역폭 효율화, 데이터 스토리지, 컴퓨팅 파워 공유 등 많은 부분에서 사용되며, 그 안에서 자원을 관리하는 대안을 제시하고 있다. 특히, P2P 파일공유 어플리케이션은 파일 교환을 위해 사용자들에게 많이 사용되어지고 있다. 대표적인 P2P 파일공유 어플리케이션으로는 eDonkey[1], Gnutella[2], Bit-torrent[3] 등의 어플리케이션들이 있다.

P2P 구조는 크게 Unstructured P2P와 Structured P2P 두 가지로 분류할 수 있다. Unstructured P2P는 전체 네트워크

※ 본 연구는 한국과학재단 특정기초연구(R01-2006-000-11167-0) 지원 및 숭실대학교 교내 연구비 지원으로 이루어졌음.
† 준 회 원 : 숭실대학교 정보통신전자공학부 석사과정
†† 정 회 원 : 숭실대학교 정보통신전자공학과 postdoc(교신저자)
††† 정 회 원 : 남서울대학교 컴퓨터학과 전임강사
†††† 정 회 원 : 숭실대학교 정보통신전자공학부 교수
논문접수 : 2008년 9월 16일
수 정 일 : 1차 2008년 8월 10일
심사완료 : 2008년 10월 30일

에 대한 정보들이 모든 피어(노드)들에 의해 관리 되거나 한 피어에게 집중되는 반면에, Structured P2P 시스템은 각각의 피어가 전체 네트워크가 아닌 부분적인 네트워크 정보를 유지 및 관리하게 하여 Unstructured P2P 시스템의 단점을 보완한 방식이다. Unstructured P2P에는 Napster[4], Gnutella, Freenet[5] 등 많은 프로토콜이 사용 되고 있다. 그렇지만, 이러한 방식에서는 Flooding 방식으로 인한 많은 트래픽 발생이 큰 문제가 되고 있다. 그러한 문제점을 고치기 위하여 Unstructured P2P 구조에서는 Hybrid P2P 방식이 제안되었다. Unstructured P2P와는 다르게, Structured P2P에서는 분산 인덱싱을 제공하는 분산 해시 테이블(DHT : Distributed Hash Table)로 컨텐츠와 피어 정보들을 공통의 단일 주소 공간으로 매핑하여 콘텐츠 저장 및 검색이 이루어지는 분산 구조의 컨텐츠-어드레싱 기반 데이터 저장 기법을 제시한다.

DHT를 사용하여 만들어진 P2P 네트워크는 하나의 오버레이 네트워크가 만들어지게 되고, 데이터 검색은 오버레이 네트워크 위에서 이루어지게 된다. 이 기법은 최대 검색 횟수 O(logN)으로 데이터 검색이 가능하기 때문에, 검색 효율에 상관없이 피어 개수를 임의로 증가 시킬 수 있다. 그렇지만 Unstructured P2P에서는 다양한 데이터의 속성 값을 이용하여 다양한 쿼리(Query)가 가능했던 반면에, 분산 해시 테이블을 사용함으로 인하여 특정 키 값만을 사용한 검색을 함으로써 쿼리가 단순화되는 단점이 있다. 이러한 DHT에서는 검색 기법이나, 관리 기법에 따라 여러 가지 어플리케이션 사례가 존재한다. CAN[6], Pastry[7], Tapestry[8], Chord[9] 등 많은 방식들이 DHT 방식으로 제안되었다.

DHT 방식 중 많이 사용되는 Chord[9, 10]는 버클리 대학에서 만들어진 방식으로, n-bit의 원형 주소 공간에 각각의 노드와 데이터의 키 값을 할당하는 방식이다. 각각의 노드와 데이터의 키 값으로부터 해싱 함수를 이용하여 주소 공간으로의 매핑이 이루어지며, 또한 주소 공간 내에 존재하는 데이터 키 k 보다 크거나 같은 노드를 k의 Successor라고 명명하여 라우팅의 효율을 높이는 데 사용한다.

본 논문에서는 기존의 Chord 및 본 연구자들이 이전에 제안한 리액티브 Chord[11]가 가지는 문제점을 분석하고, 이를 해결하기 위해 향상된 리액티브 테이블 업데이트 기법을 제안한다. 본 논문의 구성은 다음과 같다. 제 2장에서는 기존 Chord에 대한 연구방향과 Chord 및 리액티브 Chord의 문제점을 소개한다. 3장에서는 본 논문에서 제안하는 향상된 리액티브 Chord를 설명하고, 4장에서는 실험 및 토론을, 5장에서는 결론 및 향후 연구 방향을 제시한다.

2. 기존 연구

2.1 Chord 연구 동향

Chord에 관한 연구가 진행 되면서 홉 수 및 요청 지연시간을 줄이기 위한 연구는 계속되고 있고, 그 방식도 다양하게 제안되었다. 여러 개의 Chord ring을 만들어서 검색 시에 요청된 값이 보이면 바로 알려주는 방식을 사용하여 홉 수를 줄이고자 하는 방식이 제안되었다[12]. 그리고 Successor를 랜덤하게 선택하여 Finger Table을 업데이트하는 랜덤 샘플링 방식을 통해 빠르게 검색 요청에 대한 응답을 할 수 있도록 하는 방식도 제안되었다[13].

또한 Chord에서는 특정 파일에 대한 요청이 많아지는 현상이 발생하고, 그에 따른 문제점들을 해결하기 위해서 부하 분산과 관련된 연구가 많이 진행되었다. 특히, 많은 파일 전송 어플리케이션에서 사용 될 가능성이 있는 Chord에서는 이런 부분이 큰 과제로 남아 있다. 부하 분산을 위하여 많이 사용하는 방식은 캐싱을 이용한 방식이다. 그리고 다른 노드나 지역에 데이터를 복사하는 RLS(Replica Location Service) 모듈과 LRC(Local Replica Catalog) 모듈을 이용하여 부하 분산을 하는 방식도 제안되었다[14]. 또한 특정한 노드가 아닌 랜덤한 노드에 데이터를 캐싱하여 네트워크 전체의 부하 분산을 하고자 하는 연구도 진행이 되었다[15].

그리고 전체 네트워크의 환경이 유선에서 무선으로, 정적 노드에서 동적 노드로 바뀌게 됨에 따라 모바일 환경에서의 DHT 역시 많은 연구가 진행되었다. MANET에서 성공률을 높이고 라우팅을 최적화하기 위해, 라우팅 프로토콜인 AODV(Ad-hoc On-demand Distance Vector)와의 연동을 가진 연구가 진행이 되었다[16]. 그리고 모바일 환경에 따른 검색 실패를 줄이기 위해, Backtracking 방식을 사용하는 Chord도 제안되는 등 모바일 및 무선 환경이 되면서 생기는 문제점들을 생각하고 그것을 해결하기 위한 많은 방식들이 연구되었다[17].

Chord 관련 보안에서는 Sybil attack 및 라우팅 테이블 보안에 관한 연구가 진행되었다. Sybil attack이란 자신이 여러 개의 ID를 가짐으로써 다른 여러 정보들이 자신에게 올 수 있도록 만드는 방식이다. Chord의 경우는 ID를 자신이 만들기 때문에, 자신에게 여러 개의 ID를 할당 할 수 있다. 이것을 해결하기 위하여 ID를 CA(Certified Administrator)로부터 받는 방식을 제안하였다. 그리고 라우팅 테이블에 불법 사용자가 들어와 잘못 라우팅 시키는 것을 막기 위해서, 두 개의 라우팅 테이블을 사용함으로써, 하나는 빠른 검색 시에 사용을 하고 다른 하나는 빠른 검색이 실패하였을 경우 다시 라우팅을 하는 방식을 제안하였다[18].

2.2 기존 Chord

기존 Chord에서 Finger Table 업데이트를 위해서 사용하는 메시지는 크게 두 가지가 있다. 노드가 빠져 나갔음을 알기 위해 사용하는 Ping 메시지와 새로운 노드가 들어온 것을 알고 자신의 테이블을 업데이트하기 위해 주기적으로 사용하는 Find_successor 메시지가 있다. 이러한 메시지의 교환은 Stabilization 이라는 함수에 의해서 발생한다. 전자의 경우는 정해진 노드에게 하나의 메시지만을 보내서 확인을 하기 때문에 문제가 발생하지 않지만, 후자의 경우는 문제가 발생한다. Successor를 찾는 메시지를 보내기 위해서 자신은

하나의 메시지만 발생시키지만 다음 노드로 가게 되면 그 노드 역시 똑같은 과정을 반복하기 때문에, 전체 네트워크에서는 하나가 아닌 여러 개의 메시지가 발생하게 된다.

2.3 리액티브 Chord

리액티브 Chord[11]에서는 Stabilization에서 발생하는 Find_successor 메시지를 통해 주기적으로 업데이트하는 방식에서 Finger table을 데이터 요청 시에 업데이트하는 방식으로 바꾸는 것을 제안한다. 리액티브 Chord에서는 데이터 요청 메시지가 왔을 경우에만 Find_successor 메시지를 보내서 Finger table을 업데이트한다. 이전 Chord에서의 Ping 메시지는 그대로 사용을 한다. 자신이 Find_successor 메시지를 보낼 노드가 없는 상태라면, 메시지를 보내도 응답이 오지 못한다. 그렇기 때문에 노드가 살아있는지를 확인하기 위해서 Ping 메시지는 그대로 사용을 한다.

리액티브 Chord의 동작 과정은 (그림 1)과 같다.

각 동작 과정에 대해서 기술하면 다음과 같다.

단계 a) 노드는 주기적으로 자신이 유지하고 있던 Successor에게 Ping 메시지를 보내서 죽었는지 살았는지를 확인한다. 만약 죽었다면, 자신의 테이블에서 지운다.

단계 b) 다른 노드로부터 데이터 요청을 받는다.

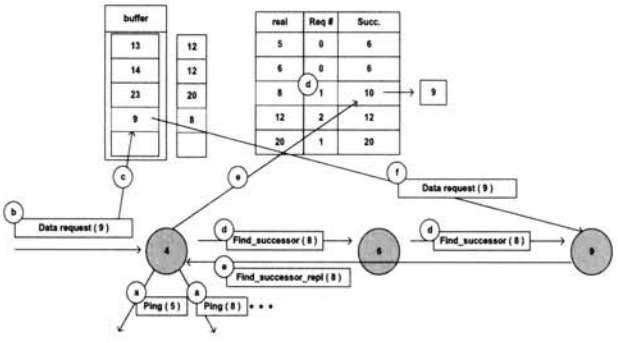
단계 c) 노드는 받은 데이터 요청 메시지를 Buffer에 넣어둔다.

단계 d) 노드는 자신의 Start 값 중에 요청 받은 메시지의 키 값보다 작은 값 중 가장 큰값을 찾는 Find_successor 메시지를 보낸다. 하지만 모든 요청에 대해서 Find_successor를 보내게 되면, 요청이 많을 시에 문제가 생길 수 있다. 그래서 제안하는 방식에서는 이전에 요청했던 것과 똑같은 것을 요청해야 하면 Find_successor 메시지를 보내지 않고 바로 Buffer에 넣는다. 한번만 보내고 나중에 안 보내면 다시 업데이트할 수 없는 문제가 생기기 때문에 요청을 보내지 않는 수를 제한 한다. n 개의 숫자를 제한하게 되면 그 요청에 대해서 n번의 요청이 오기 전 까지는 Find_successor 메시지를 한번만 보내고, n번 요청 후에 다시 보내게 된다.

단계 e) 단계 d 에서 보낸 Find_successor는 나중에 Find_successor_reply를 받는다.

단계 f) 노드는 Find_successor_reply를 받았을 때, 자신의 Finger table을 업데이트하고, 단계 c 에서 넣어 두었던 데이터 요청 메시지를 다음 노드로 전달하게 된다.

단계 g) 전달 받은 노드는 위와 같은 과정을 반복 하게 된다.



(그림 1) 리액티브 Chord의& 동작 과정

2.4 접근 방식

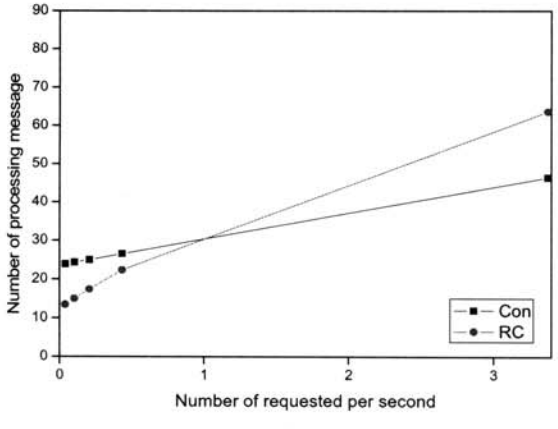
2.4.1 기존 Chord의 문제점

모바일 환경처럼 노드의 Join/Leave가 자주 발생하는 환경에서는, 요청 메시지의 성공률이 많이 떨어지게 된다. 그러한 성공률을 보장하기 위해서는 자신의 테이블을 최신 정보로 유지해야 한다. 이를 위해서는 Stabilization 주기를 빠르게 하는 방식이 있다. 그렇지만 그 주기가 빨라지게 되면 정해진 시간에 하나의 노드가 처리해야 하는 메시지의 양은 지수적으로 증가 하게 된다.[11]

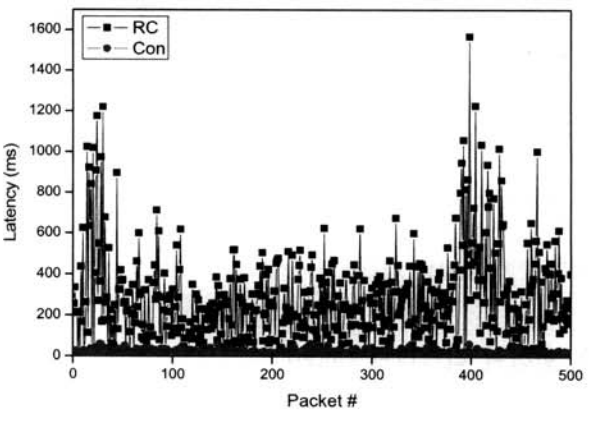
2.4.2 리액티브 Chord의 문제점

위와 같은 문제를 해결하기 위해 네트워크 트래픽을 줄이기 위한 리액티브 Chord가 제안이 되었지만, 그 역시 아래와 같은 두 가지 문제점을 가진다. 하나는 요청 메시지가 많아질 경우 트래픽의 문제와 다른 하나는 요청 응답 시간의 문제이다.

(그림 2)(a)는 노드 수가 1,000개 일 경우 초당 요청 메시지 개수에 따라 처리하는 메시지 양을 나타내는 그림이다. x축은 초당 하나

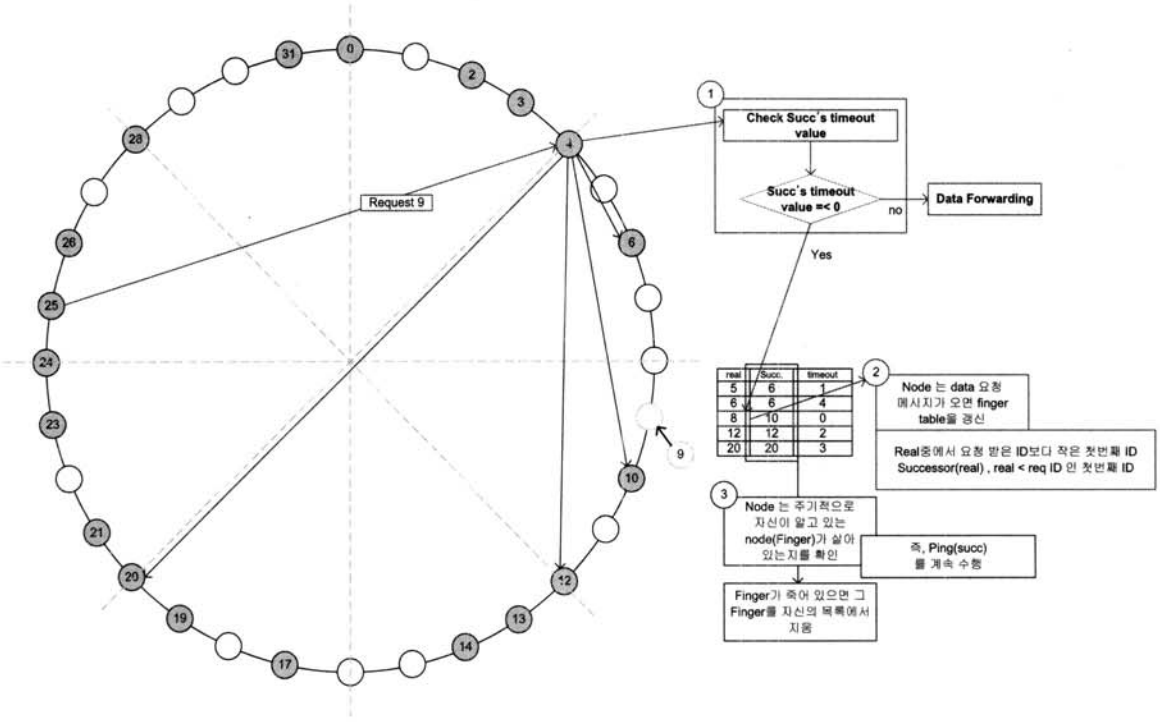


(a)



(b)

(그림 2) 리액티브 Chord에서 트래픽 문제(a)와 요청 응답 시간 문제(b)



(그림 3) 향상된 리액티브 Chord에서 Finger table 업데이트

의 노드가 보내는 요청 메시지 개수를 나타내고, y축은 노드 하나가 초당 처리한 메시지 개수를 나타낸다. 그림처럼 노드 하나가 초당 요청한 메시지의 개수가 0.9개를 넘어가게 되면, 기존의 Chord보다 노드 하나가 처리하는 메시지 개수가 많아지게 된다.

(그림 2)(b)는 1,000개의 노드가 있는 Chord 네트워크에서 500개의 메시지 요청을 하였을 때, 각 메시지의 응답 시간을 나타내는 그림이다. 그림에서 Con은 기존 코드를 나타내고, RC는 리액티브 Chord를 나타낸다. x 축은 요청 메시지의 번호를 나타내고, y 축은 각 요청 메시지의 응답 시간을 나타낸다. 그림에서 보여 주듯이 리액티브 Chord는 원래의 Chord 보다 요청 메시지의 응답 시간과 표준 편차가 굉장히 크다. 수치적으로 기존 Chord는 평균 18.43ms 정도가 걸리고 표준 편차는 11.66ms인 반면에 Reactive Chord는 평균이 289.41ms이고 표준 편차는 213.92ms이다. 이렇게 큰 요청에 대한 응답 지연 시간은 네트워크를 이용하는데 큰 불편함을 가져오게 된다.

2.4.3 본 논문의 접근 방식

본 논문에서는 Chord에서 발생하는 트래픽 문제와 리액티브 Chord에서의 단점인 큰 요청 응답 시간에 대한 문제 그리고 리액티브 Chord에서 발생하는 많은 요청에 따른 큰 트래픽 증가 문제를 해결하는 향상된 구조를 제안한다.

3. 향상된 리액티브 Chord

3.1 전체 구조

향상된 리액티브 Chord에서는 리액티브 Chord에 테이블 정보가 최신임을 알려주고 그 정보를 기반으로 테이블을 업데이트하는 알고리즘을 추가하였다. (그림 3)는 향상된 리액티브 Chord의 전체적인 그림이다.

향상된 리액티브 Chord에서는 데이터 요청 메시지가 들어오게 되면, 테이블의 정보가 최신인지 확인을 하고 그에 따라서 최신이 아니면 테이블을 업데이트하는 메시지를 보내거나 최신이면 바로 요청 메시지를 라우팅하게 된다.

3.2 동작 과정

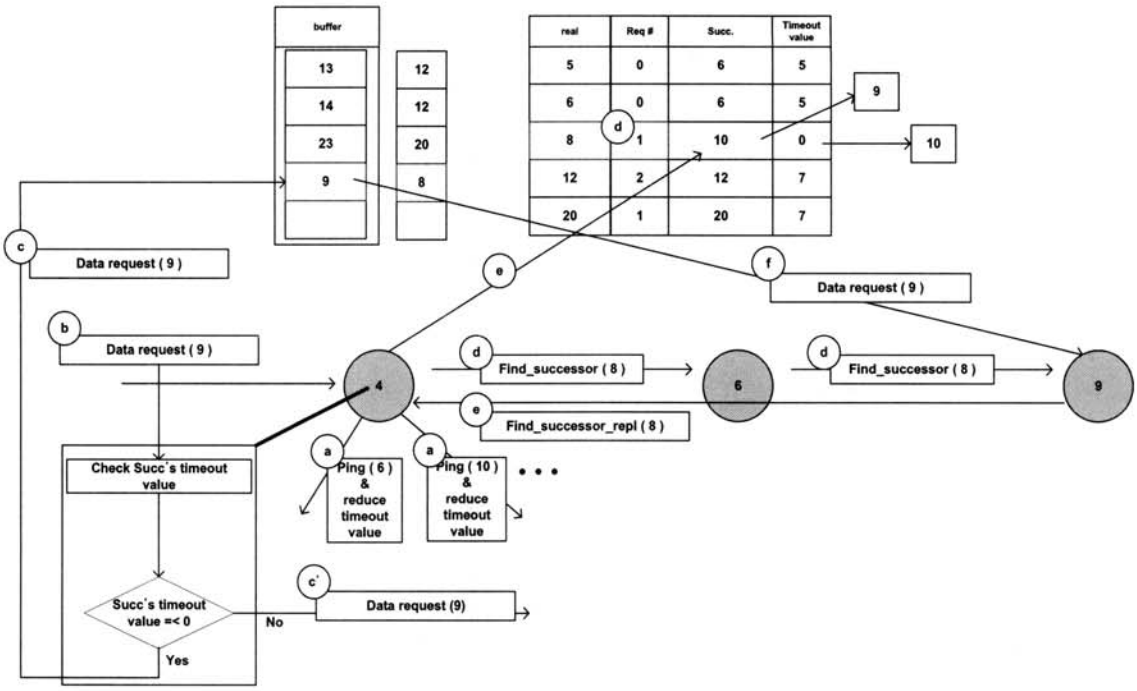
단계 a) 노드는 주기적으로 자신이 유지하고 있던 Successor에게 Ping 메시지를 보내서 죽었는지 살았는지를 확인한다. 그리고 테이블의 타임아웃 값을 감소시킨다. 만약 죽었다면, 자신의 테이블에서 지운다.

단계 b) 다른 노드로부터 데이터 요청을 받는다. 테이블의 타임아웃 값을 확인한다. 만약 타임아웃 값이 0이 되었다면 단계 c로 가고, 타임아웃 값이 0이 아니면 c′로 간다.

단계 c′) 테이블 정보가 최신 정보이므로 데이터 요청을 바로 다음 노드로 전달한다.

단계 c) 노드는 받은 데이터 요청 메시지를 Buffer에 넣어둔다.

단계 d) 노드는 자신의 Real 값 중에 요청 받은 메시지의 키 값보다 작은 값 중 가장 큰 값을 찾는 Find_successor 메시지를 보낸다. 하지만 모든 요청에 대해서 Find_successor를 보내게 되면, 요청이 많을 시에 문제가 생길 수 있다. 그래서 제안하는 방식에서는 이전에 요청했던 것과 똑같은 것을 요



(그림 4) 향상된 리액티브 Chord

청해야 하면 Find_successor 메시지를 보내지 않고 바로 Buffer에 넣는다. 한번만 보내고 나중에 안 보내면 다시 업데이트할 수 없는 문제가 생기기 때문에 요청을 보내지 않는 수를 제한한다. n개의 숫자를 제한하게 되면 그 요청에 대해서 n번의 요청이 오기 전 까지는 Find_successor 메시지를 한번만 보내고, n번 요청 후에 다시 보내게 된다. 그리고 타임아웃 값을 업데이트 한다(10으로 세팅).

단계 e) 단계 d 에서 보낸 Find_successor는 나중에 Find_successor_reply를 받는다.

단계 f) 노드는 Find_successor_reply를 받았을 때, 자신의 Finger table을 업데이트하고, 단계 c 에서 넣어두었던 데이터 요청 메시지를 다음 노드로 전달하게 된다.

단계 g) 전달 받은 노드는 위와 같은 과정을 반복 하게 된다.

4. 실험 및 토론

4.1 실험 환경

실험은 버클리에서 나온 어플리케이션인 I3에서 Chord 부분의 시뮬레이터를 통하여 실험을 수행하였다[19]. 실험에서 사용한 총 메시지의 양은 아래와 같이 계산이 되었다. 실험에서 사용한 변수 및 값은 <표 1>과 같다.

총 메시지 개수 = Ping 메시지 개수 + Pong 메시지 개수 + Stabilization 메시지 개수(옆의 노드를 찾는 것) + Stabilization 응답 메시지 개수 + Find_successor 메시지 개수 + Find_successor_Reply 메시지 개수 + 데이터 요청 메시지 개수 <표 1>에서 Stabilization 주기가 변화하면 기존 방식에서는 Ping 메시지 주기와 Find_successor 메시지 주기가 변하게 되고, 제안된 방식에서는 Ping 메시지의 주기가 변하게 된다. 이러한 변수들의 값은 다른 논문에서 자주 사용 되는 변수를 기반으로 사용하였다[20, 21]. 메시지 요청 주기는 평균 노드 하나가 초당 보내는 요청 메시지 개수를 파악하기 위하여 아래와 같이 계산 되었다.

〈표 1〉 실험에 사용된 변수 및 값

변수	값
타임 아웃	10초
노드 수	1,000노드, 2,000노드, 4,000노드
Stabilization 주기	0.25초, 0.5초, 1초(Default주기), 2초, 4초
메시지 요청 주기	0.005-3.5 (요청 / 초 / 노드)

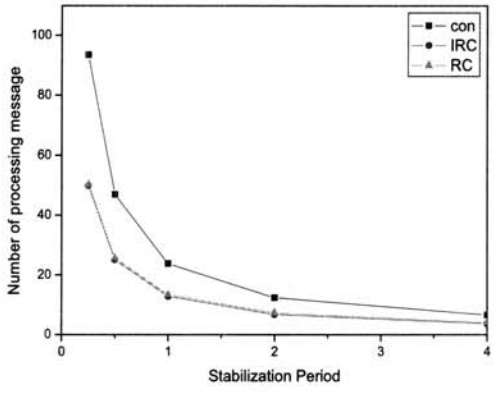
메시지 요청 주기 = (네트워크에 생성된 총 요청 메시지 개수 / 모든 요청 메시지가 처리 완료된 시간) / 총 노드 수

본 논문에서는 타임아웃 값을 10초로 설정하였다. 왜냐하면 이 값이 너무 작으면 요청에 대한 응답 지연(Latency)이 너무 커지게 되고, 또 너무 크면 요청에 대한 실패율이 커지게 되기 때문이다. 이 값을 10초로 고정한 이유는 타임아웃 값의 변화보다는 타임아웃 값이 실제로 제안된 알고리즘에서 동작하는지가 본 논문의 초점이기 때문이다.

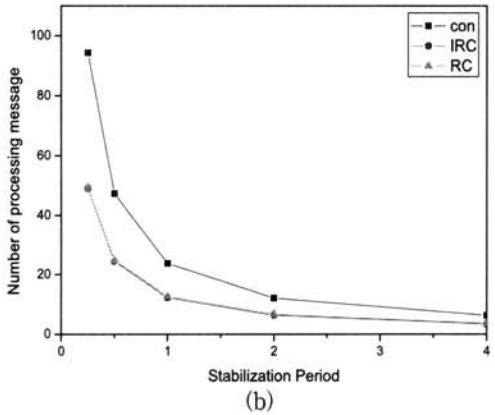
4.2 실험 결과

4.2.1 데이터 요청이 없을 경우

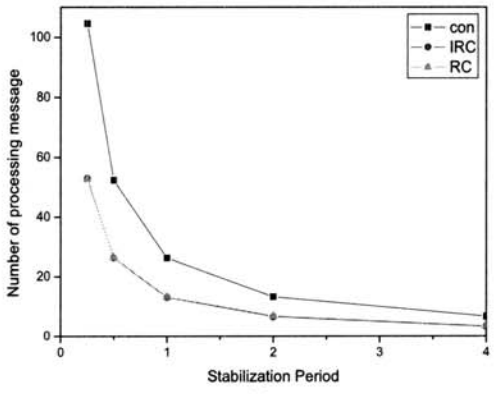
(그림 5)(a)(b)(c)는 각각 노드의 수가 1,000, 2,000, 4,000개 일 때, 데이터 요청이 없을 경우 하나의 노드에서 발생하는 메시지의 양을 보여 준다. 그림에서 Con은 기존 코드



(a)

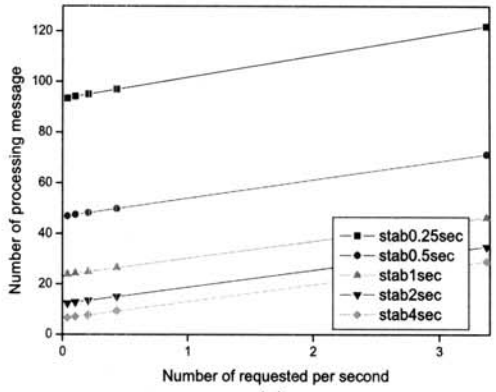


(b)

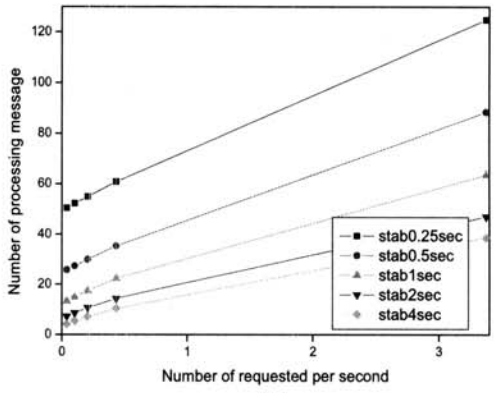


(c)

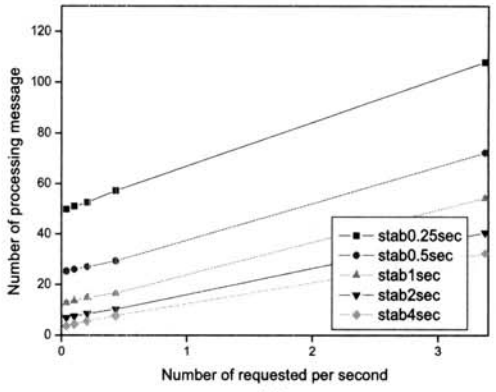
(그림 5) 노드 수가 각각 1000(a), 2000(b), 4000(c)개 일 때, 요청이 없을 경우 Stabilization 주기에 따른 노드의 메시지 처리량



(a)



(b)



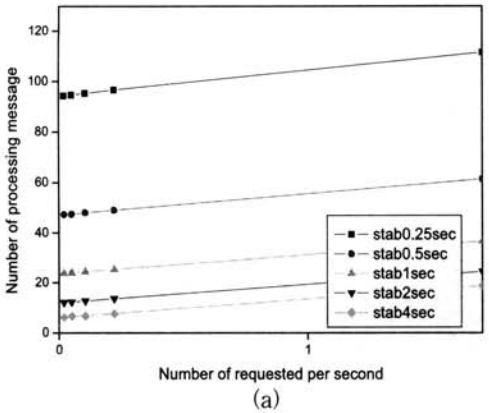
(c)

(그림 6) 기존의 Chord(a), 리액티브 Chord(b), 향상된 리액티브 Chord(c)에서 요청 메시지 개수에 따른 메시지 처리량 (1,000 노드)

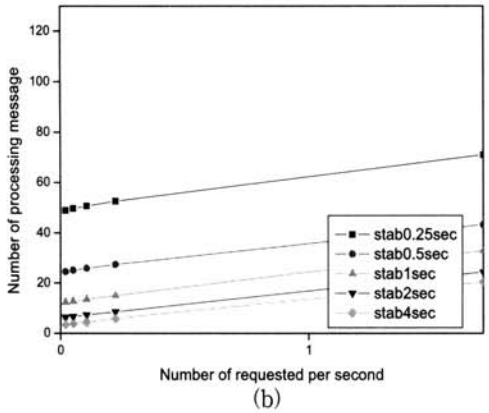
를 나타내고, RC는 리액티브 Chord 나타내며 IRC는 향상된 리액티브 Chord를 나타낸다. x 축은 Stabilization 주기이고, y 축은 하나의 노드에서 초당 발생하는 메시지의 양이다. 그림과 같이 요청이 없는 상태에서 Finger table을 유지하기 위해서 사용하는 메시지는 기존 방식이 리액티브 Chord나 향상된 리액티브 Chord에 비해 2배 정도 메시지가 많고, 리액티브 Chord와 향상된 리액티브 Chord는 사용 하는 메시지 양이 거의 같다. 예를 들어, 1000개의 노드에서 Stabilization의 주기가 1초 일 경우 기존 방식은 23개의 메시지를 처리 하지만, 리액티브 Chord 와 향상된 리액티브 Chord는 12개의 메시지만을 처리한다.

4.2.2 데이터 요청이 있을 경우

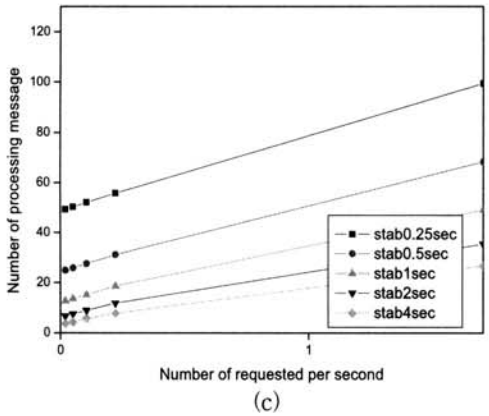
(그림 6)(a)(b)(c)는 노드 수가 총 1,000개 일 때, 각각 기존의 Chord, 리액티브 Chord, 향상된 리액티브 Chord에서 하나의 노드가 초당 보내는 메시지의 양에 따라서 하나의 노드가 처리하는 총 메시지의 개수를 보여준다. x 축은 초당 하나의 노드가 보내는 하는 데이터 요청 양을 나타내며, y 축은 초당 하나의 노드가 처리하는 메시지의 총 양이다. (그림 7)은 노드수가 2,000개, (그림 8)은 노드수가 4,000개일 경우이다. 기존의 Chord는 요청 메시지가 증가함에 따라 처리하는 메시지 양이 작은 기울기로 증가하는 반면, 리액티브 Chord 방식은 요청 메시지가 증가함에 따라 처리하는 메



(a)

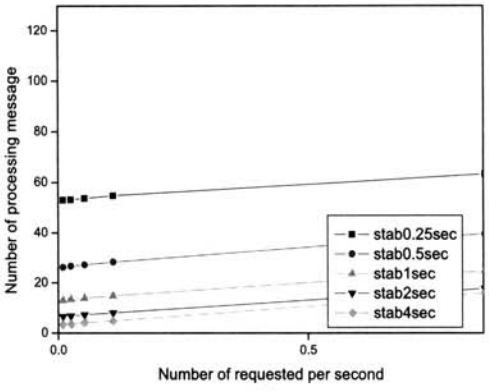


(b)

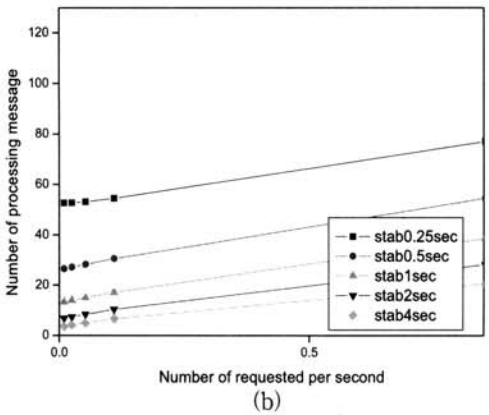


(c)

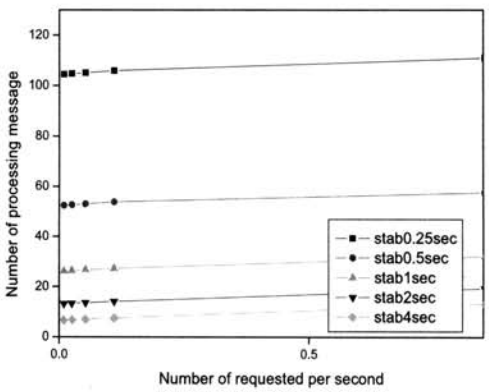
(그림 7) 기존의 Chord(a), 리액티브 Chord(b), 향상된 리액티브 Chord(c)에서 요청 메시지 개수에 따른 메시지 처리량 (2,000 노드)



(a)



(b)



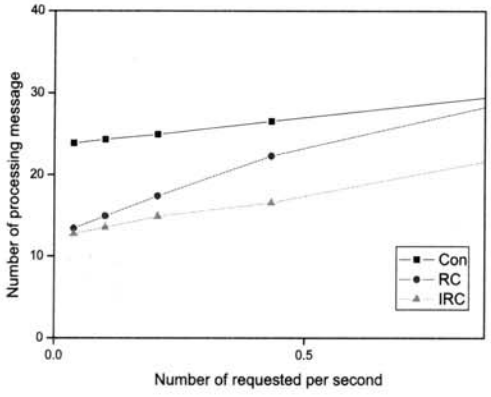
(c)

(그림 8) 기존의 Chord(a), 리액티브 Chord(b), 향상된 리액티브 Chord(c)에서 요청 메시지 개수에 따른 메시지 처리량 (4,000 노드)

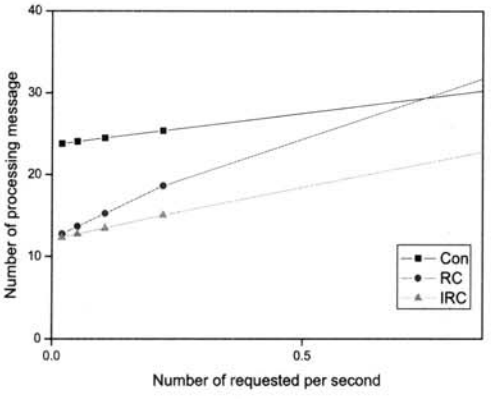
시지 양이 기존 방식보다 큰 기울기로 증가하게 된다. 향상된 리액티브 Chord 역시 요청 메시지가 증가함에 따라 처리하는 메시지의 양이 증가하기는 하지만, 리액티브 Chord에 비해서는 크게 좋아졌다. 기존의 Chord, 리액티브 Chord, 향상된 Reactive Chord의 기울기 값을 비교해보면 노드수가 1,000개 일 경우, 각각 5.92, 19.09, 10.18이다. 1,000개의 노드를 기준으로 하고, 기존 방식에서 Stabilization 주기를 default 값인 1로 놓고 본다면 리액티브 Chord는 노드가 초당 0.9개 이상의 메시지 요청을 보낼 경우 기존의 방식보다 많은 트래픽을 발생 하는 문제가 생기지만, 향상된 리액티브 Chord의 경우에는 하나의 노드가 초당 2.4개의 요청을 보낼 경우 기존의 Chord 방식보다 많은 트래픽을 발생 시킨다. 그렇지만, P2P 관련 통계 자료들을 본다면 보통 하나의 노드가 초당 데이터를 요청하는 개수는 0.02-0.2개 이다[22, 23]. 즉, 보통의 평범한 환경에서는 기존 방식 보다 적은 트래픽이 발생하게 되며, 모바일 환경을 가정하여 기존 방식에서 Stabilization 주기를 빠르게 한다면, 더 좋은 결과를 얻을 수 있다.

4.2.3 정량적 비교

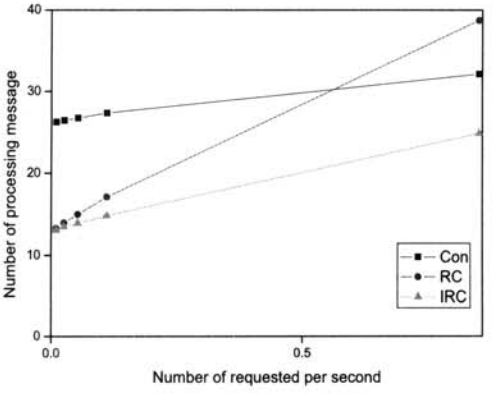
(그림 9)(a)(b)(c)는 Stabilization 주기를 1초로 하고, 각각 노드 수가 1,000, 2,000, 4,000 개 일 때 성능 비교 값을 보여



(a)



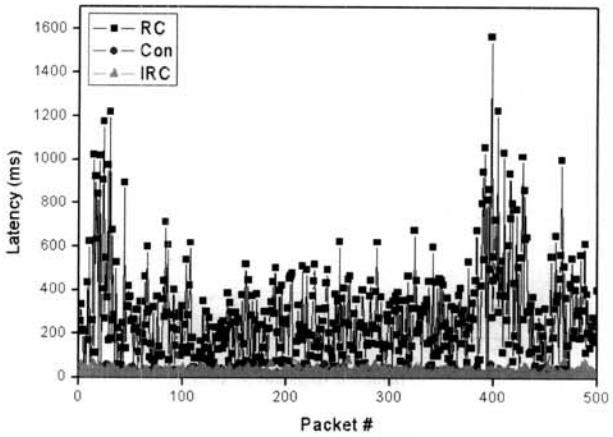
(b)



(c)

(그림 9) Stabilization 주기가 1초이고, 노드수가 1,000(a), 2,000(b), 4,000(c) 개 일 때, 기존의 Chord, 리액티브 Chord 및 향상된 리액티브 Chord의 메시지 처리량

준다. 그림을 보면 향상된 리액티브 Chord가 리액티브 Chord에 비해 기울기가 많이 줄어들었음을 확인할 수 있다. 리액티브 Chord의 경우 기존의 Chord보다 트래픽양이 많아지는 지점(초당 요청 수)이 노드수가 많아질수록 크게 낮아지지만(왼쪽으로 이동), 향상된 리액티브 Chord의 경우는 거의 변화가 없다. 예를 들어, 노드 수가 1,000개일 경우 기존의 Chord와 리액티브 Chord가 만나는 점은 1을 조금 넘지만, 향상된 리액티브 Chord와 만나는 점은 2.5에 가깝다. 2,000, 4,000개의 노드일 경우도 리액티브 Chord에 비해서



(그림 10) 요청 메시지의 요청 응답 지연 시간

향상된 리액티브 Chord가 기존의 Chord와 만나는 점은 뒤에 있다.

4.2.4 요청 응답 지연 시간

(그림 10)는 노드수가 1,000개 일 때, 임의로 500개의 메시지를 뽑아 각각의 응답 시간을 비교한 그림이다. 리액티브 Chord는 기존 Chord 방식에 비해서 요청 응답 지연 시간이 굉장히 큰 반면에 향상된 리액티브 Chord는 기존의 Chord 방식과 거의 같은 응답 지연 시간을 가지고 있다. 각각의 평균과 표준 편차를 비교해 보면, 기존의 Chord는 평균이 18.43ms, 표준 편차가 11.66ms 이고, 리액티브 Chord는 평균 289.41ms, 표준 편차 213.92ms 이며, 향상된 리액티브 Chord는 평균 18ms, 표준 편차 11.02ms 이다.

5. 결 론

본 논문에서는, DHT를 위해 제안된 Chord의 문제점 및 이의 문제점을 해결하기 위해 제안 되었던 리액티브 Chord의 문제점을 지적하고, 이를 개선한 향상된 리액티브 Chord를 제안하였다. 리액티브 Chord는 기존 Chord에서 발생할 수 있는 트래픽 문제에 대해서 해결하였지만, 요청 응답 시간 지연 및 요청 메시지 증가에 따른 트래픽 증가의 문제가 있었다.

향상된 리액티브 Chord는 테이블의 정보가 최신 정보인지를 확인하고, 그에 따라서 메시지를 바로 라우팅 할 것인지, 테이블을 업데이트 할 것인지를 정하게 된다. 이를 통해 필요치 않은 테이블 업데이트 메시지를 줄이게 되었고, 리액티브 Chord에서의 문제였던 요청 메시지 증가에 따른 트래픽 증가 문제를 약간이나마 해결하였다. 제안된 방식은 실험을 통하여 요청에 대한 응답 시간의 지연은 기존의 Chord 방식까지 따라가게 되었고 메시지 처리량 역시 리액티브 Chord에 비해 줄었음을 확인하였다. 하나의 노드가 처리하는 메시지가 줄어들게 되는 것은 네트워크 전체에서 차지하는 트래픽양이 줄어드는 것을 의미한다.

제안된 향상된 리액티브 Chord 방식의 장점은, 모바일 환

경의 P2P에서 기존의 방식보다 Finger table을 업데이트하는데 사용하는 메시지의 양을 줄여서 전체 P2P 네트워크의 트래픽을 줄일 수 있는 것이다. 기존 유선 방식의 P2P에서는 하나의 노드가 처리하는 메시지의 양이 많지 않아서 크게 문제가 되지 않을 것이라는 생각되었지만, 모바일 환경이 되게 되면, 네트워크를 유지하기 위해서 발생시켜야 하는 메시지가 많아질 수 있기 때문이다. 제안된 방식의 단점은 요청 메시지가 많아지게 되면 개선되기 이전의 방식처럼 그 정도는 작지만 네트워크의 트래픽이 증가한다는 것이다. 그렇지만, 그러한 부분은 데이터 요청 메시지를 받았을 때, 보내는 Find_successor 메시지를 보내는 정도를 줄이거나, 타임아웃 값을 조절하면 문제를 해결할 수 있으리라 생각한다.

향후 연구 방향으로는, 데이터 요청에 따라서 증가하는 Find_successor 메시지 양에 대한 단점을 극복하기 위하여 Adaptive Chord를 개발하려 한다. Adaptive Chord란 데이터 요청의 증감에 따라 데이터 요청 메시지가 많아지면 기존 Chord를 사용하고, 데이터 요청 메시지가 작아지면 향상된 리액티브 Chord를 사용하는 방식이다. 해당 방식을 사용하게 되면 데이터 요청의 증감에 무관하게 네트워크의 전체 트래픽을 일정하게 유지할 수 있을 것으로 사료된다. 또한 요청 지연에 사용된 최적의 타임아웃 값을 찾으려 한다. 최적의 타임아웃 값이란 요청 지연과 실패율 최소로 하는 값으로 다양한 실험을 통해 찾을 수 있을 것으로 사료된다.

참 고 문 헌

[1] eDonkey, "http://www.donkeyp2p.com/".

[2] Gnutella. "http://www.gnutella.com/".

[3] Bittorrent, "http://bitconjurer.org/BitTorrent/".

[4] Napster. "http://www.napster.com/".

[5] I. Clarke et al., Freenet: A Distributed Anonymous Information Storage and Retrieval System, available at http://freenetproject.org/freenet.pdf, 1999.

[6] S. Ratnasamy et al., "A Scalable Content Addressable Network," Proc. ACM SIGCOMM, pp.161-72, 2001.

[7] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," In IFIP/ACM Int'l Conf. on Distributed Systems Platforms(Middleware), pp.329-350, Nov., 2001.

[8] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz, "Tapestry: A Resilient Global-scale Overlay for Service Deployment," IEEE Journal on Selected Areas in Communications, Vol.22, No.1, pp.41-43, 2004.

[9] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications", In IEEE/ACM Transactions on Networking, Vol.12, No.2, pp. 205-218, Apr., 2004.

[10] H. Park, K. Park, "P2P Technology Trend and Application to Home Network", 전자통신동향분석, Vol.21, No.5, Oct 2006.

[11] 윤영효, 곽후근, 김정길, 정규식, "모바일 환경을 위한 리액티브 방식의 Chord", KCC 2008, 한국정보과학회, 2007.

[12] P. Flocchini and A. Nayak, "Enhancing Peer-to-Peer Systems Through Redundancy", IEEE Journal on Selected Areas in Communications, Vol.25, No.1, pp.15-24, Jan., 2007.

[13] Z. Hui, A. Goel, and R. Govindan, "Improving lookup latency in distributed hash table systems using random sampling", IEEE/ACM Transactions on Networking, Vol.13, pp.1121-1134, Nov., 2005.

[14] M. Cai, A. Chervenak, and M. Frank, "A Peer-to-Peer Replica Location Service Based on a Distributed Hash Table", Supercomputing, 2004. Proceedings of the ACM/IEEE SC2004 Conference, pp.56-61, July, 2004.

[15] P. Keleher, S. Bhattacharjee, and B. Silaghi, "Are Virtualized Overlay Networks Too Much of a Good Thing?", Proceedings of International Workshop on Peer-to-Peer Systems (IPTPS'02), pp.225-231, Mar. 2002

[16] M. Dischinger, "Mobility Enhancements to an Approach for Structured Overlay Routing in Wireless Mobile Ad Hoc Networks", Diploma Thesis, System Architecture Group, University of Karlsruhe, Nov., 2005.

[17] S. Lee and J. Jang, "Backtracking Chord over Mobile Ad-hoc Network", 한국정보과학회 춘계학술대회, pp.517-519, 2004.

[18] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D.S. Wallach, "Secure routing for structured peer-to-peer overlay networks" , Massachusetts (2002) To appear Boston, 2002.

[19] I. Stoicam, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure", ACM SIGCOMM Computer Communication Review, Vol.32, pp.73-86, 2002.

[20] C. Cramer and T. Fuhrmann, "Performance Evaluation of Chord in Mobile Ad Hoc Networks", ACM International Conference on Mobile Computing and Networking, pp.48-53, 2006.

[21] C. Tang, M. Buco, and R. Chang, "Low traffic overlay networks with large routing tables", ACM Special Interest Group on Measurement and Evaluation, Vol.33, pp.14-25, 2005.

[22] W. Acosta and S. Chandra, "Trace Driven Analysis of the Long Term Evolution of Gnutella Peer-to-Peer Traffic", Lecture Notes in Computer Science, Vol.4427, pp.42-51, 2007.

[23] K. Gummadi, R. Dunn, and S. Saroiu, "A Measurement Study of Napster and Gnutella as Examples of Peer-to-Peer File Sharing Systems", Symposium on Operating Systems Principles, Oct., 2003.

윤 영 효

e-mail : yyhpower@q.ssu.ac.kr
2006년 숭실대학교 정보통신전자공학부 (학사)
2006년 3월~현 재 숭실대학교 정보통신전자공학부 석사과정
관심분야 : 네트워크 컴퓨팅 및 보안

곽 후 근

e-mail : gobarian@q.ssu.ac.kr
1996년 호서대학교 전자공학과(학사)
1998년 숭실대학교 전자공학과대학원 (석사)
1998년~2006 숭실대학교 전자공학과 대학원(박사)
1998년 8월~2000년 7월 (주) 3R 부설 연구소 주임 연구원
2006년 3월~현 재 숭실대학교 정보통신전자공학과 대학원 postdoc
관심분야 : 네트워크 컴퓨팅 및 보안

김 정 길

e-mail : cgkim@nsu.ac.kr
2003년 8월 연세대학교 컴퓨터학과(석사)
2006년 8월 연세대학교 컴퓨터학과 (공학박사)
2006년 9월~2007년 8월 연세대학교 컴퓨터학과 박사후 연구원
2007년 9월~2008년 2월 연세대학교 컴퓨터학과 연구교수
2008년 3월~현 재 남서울대학교 컴퓨터학과 전임강사
연구분야 : 멀티미디어 임베디드 시스템, 컴퓨터구조

정 규 식

e-mail : kchung@q.ssu.ac.kr
1979년 서울대학교 전자공학과(공학사)
1981년 한국과학기술원 전산학과(이학석사)
1986년 미국 Univ. of Southern California (컴퓨터공학석사)
1990년 미국 Univ. of Southern California (컴퓨터공학박사)
1998년 2월~1999년 2월 미국 IBM Almaden 연구소 방문 연구원
1990년 9월~현 재 숭실대학교 정보통신전자공학부 교수
관심분야 : 네트워크 컴퓨팅 및 보안