# 분산 해쉬 테이블 기반 피어 투 피어 컴퓨팅 시스템에서 가용성 향상 객체 복제 기법

손영성* · 정일동** · 김경석***

## Object Replication Methods for High Availability in DHT based P2P Computing System

Young-Sung Son* · Il-dong Jung** · Kyongsok Kim***

요 약

최근 인터넷과 PC의 발달로 인터넷 환경에서 대규모 분산 컴퓨팅 환경을 구성하는 적절한 기술로 P2P 네트워크가 소개되어 mp3 파일 공유와 같은 응용 분야에 널리 쓰이고 있다. P2P 네트워크에서 가장 중요한 기능인 데이터를 위치시키고 (location) 탐색하기 위해서 분산 해쉬 테이블(DHT:Distributed Hash Table)을 이용한다. 본 논문에는 정열 원칙을 제공하여 간단하면서도 효과적인 방법으로 상호 연결 및 검색을 제공하는 MagicSquare에서 자원의 복제 및 공유를 통해서 컴퓨팅 인프라 전반에 걸친 신뢰성과 결함감내 능력을 향상시키는 복제기술을 소개한다. 현재까지의 P2P 네트워크는 복제된 데이터의 일관성 유지를 위해 노드들 간에 필요한 통신 매커니즘에 대한 고려가 부족하다. 이를 위해서 본 논문은 자원의 복제를 통한 신뢰성 향상 기법에 대해서 소개한다. 마지막으로 시뮬레이션을 통해서 MagicSquare의 성능을 검증하였다.

ABSTRACT

Recently, there are many researches for P2P network. P2P network technologies are a good solution about making a wide spread distributed computing in the internet environment. The recent algorithms developed by several research groups for the lookup problem present a simple and general interface, a distributed hash table (DHT). In this paper, we also introduce new schemes that enhance the available rate of an object in the MagicSquare network. Replication scheme is to replicate an object with many replicas and save them to several nodes. Fragmentation scheme is to divide an object into several fragments and save them to several nodes. Replicated Fragmentation scheme is to mix replication scheme and fragmentation scheme. We will show result of simulation for our the proposed scheme.

키워드

P2P 네트워크, 분산 해쉬 테이블, 유비쿼터스 네트워크

## I. 서 론

인터넷과 PC의 발달은 정보의 분산과 공유를 가속화하였으며, 컴퓨팅 구조가 인터넷 기반으로 변화고 있다. 월드와이드웹(WWW)으로 대표되는 웹 컴퓨팅 환경은 단순한 클라이언트-서버 컴퓨팅 환경에서 대단위 분산 다중 서버로 구성되는 환경으로 바뀌었다. 또한 전세계의 사용자들이 만들어내는 엄청난 정보는 웹과 데이터베이스에 축척되고 있다. 이러한 정보들은 서버뿐만 아니라 개인 PC및 다양한 미디어에까지 축척되어, 이는 검

* 한국전자통신연구원 디지털홈연구단
** LG전자 디지털어플라이언스 사업본부 연구소
*** 부산대학교 컴퓨터공학과

접수일자 : 2007. 6. 19

색 포털의 한계, 즉 정보 탐색 범위 제한, 부정확성, 개인들의 검색에 대한 욕구 불만 등의 문제로 사람들은 다른 대안을 찾기 시작했다. 사용자 스스로 개인의 컴퓨터 자원과 인적 자원을 공공의 목적을 위해서 제공하고 이를 전체적으로 연계하여 개인의 피어(컴퓨터) 사이에 서로 공유하는 P2P (Peer-to-Peer) 컴퓨팅이 현실화되었다 [1, 2].

P2P 컴퓨팅은 다수의 자원을 효율적으로 사용하기 위하여 다음과 같은 요소 기술이 필요하다. 피어 및 자원의 이름 정의 방법(naming), 시스템 구조체 구성 방법(constructing), 자원 탐색 방법(searching), 자원 접근의 가용성을 높이기 위한 중복 저장 방법 (redundant storing), 자원 저장 위치 선정 방법 (placement), 자원 저장 영구성 (permanence), 인접 피어 선택(selection of nearby servers), 익명성(anonymity), 인증(authentication) 등과 같은 요소 기술에 대한 연구가 진행 중이다 [3].

본 논문에서는 기존의 P2P 시스템에서 고려하고 있지 못한 피어의 능력을 고려한 MagicSquare P2P 프로토콜을 소개하고, 이를 다양한 응용으로 이용되기 위해서 필수로 요구되는 가용성 향상 기법을 보인다. 이를 통해서 P2P 시스템이 다양한 응용에 적용 가능함을 제시한다.

2장에서는 관련 연구를 살펴보고, 3장에서는 본 논문에서 제안하는 P2P 프로토콜인 MagicSquare 를 소개한다. 4장에서는 복제기법을 이용하여 P2P 네트워크의 가용성을 향상시키는 방법을 소개한다. 5장에서는 시뮬레이션을 통해서 제안한 프로토콜의 성능을 검증한다. 마지막으로 6장에서는 결론과 향후 연구를 제시하고 논문을 마친다.

## II. 관련 연구

### 2.1 Skiplist

스킵 리스트는 임의성 기반의 균형 트리이다. 이진 트리(binary tree) 가 특정 상황에서 성능이 떨어지는 문제를 보완하여 균형 트리(balanced tree) 가 제시되었다. 균형 트리는 트리의 균형 조건을 유지하기 위해 트리를 재구성하는 연산을 하는데, 트리가 커짐에 따라 트리의 재구성 연산의 시간 비용이 증가한다.

스킵 리스트는 랜덤 수 생성기를 사용하여 균형 트리를 확률적으로 변형한 형태이다. 스킵 리스트가 가장 나쁜 성능을 보이는 경우가 있더라도 이진 트리처럼 입력되는 순서가 성능에 영향을 미치는 경우는 없다. 스킵 리스트의 균형을 이루는 속성은 탐색 트리에 랜덤 삽입 (insertion) 을 하여 균형을 이루는 속성과 비슷하다.

확률적으로 균형을 만드는 것이 명시적으로 균형을 유지하는 연산을 하는 것보다 쉬우며, 알고리즘도 더 간단하다. 스킵 리스트의 알고리즘이 단순하기 때문에 구현하기도 쉽고, 균형 트리보다 속도도 더 빠르다. 또한 스킵 리스트는 공간 효율성이 아주 높다.

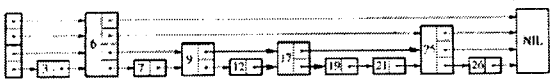



그림 1. 스킵리스트의 구조
Fig 1. Skiplist structure

스킵리스트는 기존의 연결 리스트(linked list) 에 임의의 노드마다 몇 개의 연결을 추가한 것으로 그림 1 과 같은 모양을 이룬다. 각 노드가 가지는 연결의 수를 레벨(level) 이라 하며, 이 레벨은 랜덤하게 생성하기 때문에 균형을 이루며 임의의 노드를 찾는데 O(logN) 의 시간 복잡도(time complexity) 를 가진다 [4].

### 2.2 P2P(오버레이 네트워크) 의 라우팅

오버레이 네트워크(Overlay Network, =P2P 네트워크) 분야에서 사용하는 라우팅의 의미는 일반적인 네트워크 분야에서 사용하는 라우팅의 의미와 달리, 피어가 가진 다른 피어들의 리스트 중에서 목적지와 가장 가깝거나 효율적으로 전달할 것 같은 피어에게 메시지를 전달하는 것을 말한다.

Chord는 정확성(correct) 과 손끝 테이블(finger table) 을 사용해서 소스 피어와 ID 영역의 거리가 1/2이 되는 피어, 1/4, 1/8, 2의 지수승의 거리에 있는 피어의 IP를 알아낸다. 이 구조는 스킵 리스트 데이터 구조와 비슷하다. 피어는 키 K를 요청하는 질의를 손끝 테이블에서 K를 넘지 않는 가장 큰 ID의 피어 (K의 계승자(successor)) 에 전달한다. 손끝 테이블이 2의 지수 구조로 되어 있기 때문에 메시지를 전달하는 피어와 K까지의 ID 공간의 1/2을 전달할 수 있다. 따라서 질의를 처리하는데 O(log N) 메시지가 필요하다 [5]. Chord의 손끝 테이블과 라우팅 경로는 그림 2 와 같다.

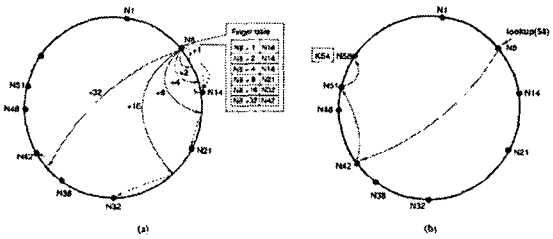


그림 2. (a) 피어 8 의 손끝 테이블 모양
(b) 피어 8 에서 K54 를 질의했을 경우의 라우팅 경로
Fig 2. (a) finger table of peer 8
(b) routing path for querying K54 of peer 8

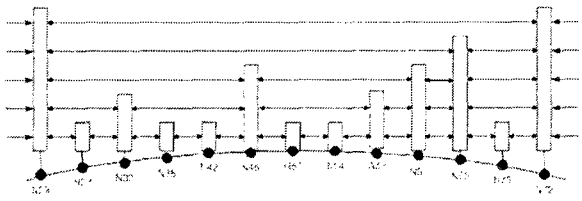


그림 3. MagicSquare 의 구조(전체 ID공간 중 일부분). N23 과 N72 의 네트워크 성능이 가장 우수한 집합에 포함되며, 높이는 5라고 가정한다.
Fig 3. MagicSquare structure between N23 and N72.

## III. MagicSquare 프로토콜

마방진은 가로, 세로, 및 대각선에 있는 각각의 합이 같도록 배열한 입방체를 말한다. 마방진에서는 각각의 숫자와 그 배열은 특별한 의미가 없지만 각 수를 적절히 배치하여, 완성된 마방진에서는 어느 방향에서 보아도 보이는 면의 모든 수를 합하면 같은 합을 가진다. 이처럼 본 논문에서 제안한 MagicSquare 에서도 각 피어가 가지는 정보는 의미가 없더라도 전체 시스템에서는 안정적인 구조를 형성한다는 의미를 가지고, 어떤 피어에서 질의를 하더라도 비슷한 성능이 나올 수 있도록 설계하였기 때문에 MagicSquare 시스템을 구축하는데 있어 마방진을 구축하는 알고리즘을 적용한 것은 아니지만 상징적인 의미로 MagicSquare(마방진) 이라고 이름을 지었다.

본 장에서는 피어의 컴퓨팅 능력을 고려한 분산 해쉬 테이블 기반 피어 투 피어 컴퓨팅 시스템인 MagicSquare 의 기본 프로토콜에 대해서 설명한다.

### 3.1 시스템 구조

P2P 시스템에서는 피어의 접속과 떠남, 데이터의 탐색이 중요한 연산이지만, 이 모든 연산을 관장하는 분산 해쉬 테이블을 생성하고 사용하는 방법은 P2P 시스템의 구조에 영향을 받는다. 이 절에서는 피어의 탐색 비용을 O(logN) 으로 유지하면서 피어의 성능을 반영하여 성능을 향상할 수 있는 MagicSquare 의 구조에 대해서 살펴본다.

MagicSquare 는 그림 3 과 같이 스킵리스트와 비슷한 모양을 가진다. 스킵리스트의 각 노드는 MagicSquare 의 피어에 대응되며, 링크는 피어 사이의 연결에 대응된다. 피어 사이의 연결 정보와 다른 피어의 정보는 해당 피어의 라우팅 테이블에 저장한다. 그러므로 시스템에서 가장 많은 연결을 가지는 피어는 높이가 가장 높은 피어이며, 2*높이 개의 피어 정보를 관리한다. 보기를 들어, N46 피어는 시스템에서 6개의 피어와 연결을 가지므로, N46이 관리하는 라우팅 테이블에는 {N23, N32, N42, N51, N57, N61}을 저장한다. N61 피어는 {N57, N61}을 저장한다. N61과 같이 경우에 따라 높이는 높지만 연결을 적게 가질 수 있지만, 전체 분포에는 큰 영향을 미치지 못하기 때문에 성능 저하는 없을 것으로 본다.

MagicSquare 는 피어의 가용 네트워크 대역폭을 반영하여 피어의 높이를 정한다. 그림 3 의 각 피어는 높이를 스스로 결정하는 것이 아니고, 자신의 가용 네트워크 대역폭을 측정하여 결정한다. MagicSquare 는 피어의 높이가 높은 피어 집합이 메시지 라우팅에 자주 참여하는데, 이 피어 집합의 네트워크 성능이 우수하기 때문에 메시지 라우팅 성능이 향상될 것이다. 결과적으로 MagicSquare 의 가용 네트워크 성능이 우수한 피어 집합이 시스템의 성능에 결정적인 역할을 한다.

### 3.2 데이터 찾기

피어가 MagicSquare 에 접속하기 위해서 피어의 위치를 찾는 연산을 수행하는데, 이 연산은 데이터의 담당자를 찾는 연산과 같다. 피어가 MagicSquare 에 접속하는 절차를 살펴보기에 앞서 데이터의 담당자를 찾는 연산에 대해서 먼저 살펴본다.

P2P 컴퓨팅을 이용한 인터넷 파일 시스템에서는 피어가 시스템에 저장된 파일에 대한 정보를 모두 가지고 있지 않다. 피어가 시스템에 저장된 데이터를 찾기 위해서는 데이터의 담당자를 찾는 방법을 알고 있으면 된다. MagicSquare 에 참여하는 피어는 ID 와 높이에 따라 연결되는 피어가 정해지며, 연결을 유지하기 위해서 라우팅 테이블에 정보를 유지한다.

데이터를 요청하는 피어는 자신이 관리하는 라우팅 테이블에서 목적 데이터를 저장한 피어에 가장 가까운 피어를 선택해서 메시지를 전달하면, 메시지를 전달받은 피어도 역시 자신이 관리하는 라우팅 테이블에서 목적 데이터를 저장한 피어에 가장 가까운 피어를 선택하여 메시지를 전달한다. 이 과정을 반복하여 데이터를 저장하고 있는 피어에 연결한다. 이처럼 데이터를 요청하는 질의는 데이터의 담당자를 만날 때까지 ID 공간을 따라 전달된다. ID 공간에서 데이터의 담당자를 찾는 알고리즘은 그림 4 과 같다. 그림 4 에서 피어의 ID 가 앞에 붙어 있는 것은 리모트 호출과 리모트 변수 사용을 의미하고, 피어의 ID 가 생략된 것은 지역 변수와 지역 함수 호출을 의미한다.

그림 4 의 find_manager()는 데이터의 담당자가 인접했으면 담당자를 반환하고, 그렇지 않은 경우에는 현재 피어의 라우팅 테이블에서 담당자와 1) 시계 반대 방향으로 가장 가까운 피어와 2) 시계 방향으로 가장 가까운 피어를 선택하여 메시지를 동시에 전달한다. 메시지에는 최종 전송한 피어의 ID가 기록되어 있으므로 같은 피어에 다시 전달되지 않는다. 양방향으로 전달하면 메시지가 많이 생성되지만, 높이가 높은 (네트워크 성능이 좋은) 피어를 만나서 데이터의 담당자를 빨리 찾을 가능성이 커지기 때문에 시스템의 응답 시간이 빨라질 것으로 보인다.

closest_peers()는 자신이 관리하는 라우팅 테이블에서 데이터의 담당자와 가장 가까울 것으로 예상되는 두 개의 피어를 반환한다. 스킵리스트에서 높이가 낮은 노드를 선택하는 것보다 높이가 높은 노드를 선택했을 때의 검색의 진행이 더 빠르다. 따라서 MagicSquare 에서도 스킵리스트처럼 데이터의 검색 속도를 높이기 위해서 피어가 진행 방향으로 담당자를 지나치지 않으면서 높이가 높은 피어를 선택한다.

```
n.find_manager(k)  // manager(k)를 찾을 때
{
  if k ∈ n (predecessor, n] then
    return n;
  else if k ∈ (n, successor] then
    return successor;
  else
    n' = closest_peers(k);
    // 병렬로 실행
    for(i=0;i<n'.length;i++)
      make_thread(return n'[i].find_manager(k));
}
n.closest_peers(k)
{
  do
  {
// 0번째: 시계 반대 방향 피어 중 가장 높이가 높은 피어
// 1번째: 시계 방향 피어 중 가장 높이가 높은 피어
    highest_peers = find_highest_peers(n.route_table);
  } while( highest_peers 가 k를 지남 )
    // k를 지나지 않으면 루프를 빠져나옴
  return highest_peers;
}
```

그림 4. 키의 담당자를 찾는 의사 (Pseudo) 코드
Fig 4. Pseudo code of finding manager for key

그림 5 은 MagicSquare 에서 K53 의 담당자를 찾기 위해서 메시지를 라우팅하는 경로를 보여준다. 보기에서는 K53 의 담당자는 N54 이며, 높이가 높은 피어를 선택해서 탐색을 진행하다가 메시지가 담당자를 지나치지 않도록 피어를 선택하는 과정이 스킵리스트의 데이터 검색과 비슷함을 알 수 있다.

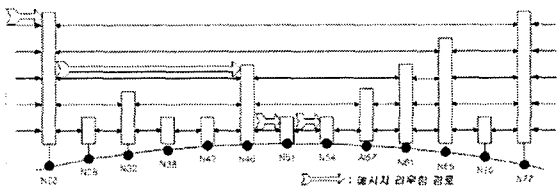


그림 5. MagicSquare 에서 manager(K53)을 찾기 위해서 메시지가 라우팅하는 경로
Fig 5. Message routing path for finding manager(K53) in MagicSquare

피어는 스킵 리스트를 사용해서 라우팅 테이블을 관리하기 때문에 피어를 찾는 연산은 분산된 스킵 리스트 조각을 이용해서 특정 값을 찾는 연산과 같다. 분산된 스킵 리스트 조각을 사용하더라도 특정 값을 찾을 때 레벨이 높은 노드 (리스트 안의 노드)를 사용해서 진행하기 때문에 비용은 O(log N)에 근접할 것이다.

## Ⅳ. 복제를 통한 가용성 향상 기법

분산 해쉬 테이블 기반 파일 저장 방법은 P2P 시스템의 분산 구조, 자가 구성, 피어 오류 반경 제한 등의 기본 사상을 분산 저장 방법과 접목하여 대용량 분산 저장 시스템을 만들기 위한 필수적인 기술이다. 본 장에서는 본 논문에서 제안하는 분산 해쉬 테이블 기반 P2P 시스템인 MagicSquare 에서의 파일 저장 방법에 대해서 설명한다.

### 4.1 분산 해쉬 테이블 기반 시스템에서의 파일 저장 방법의 문제점

기존 P2P 시스템의 파일 저장 및 검색 방법은 다음과 같다. 각 피어는 전체 시스템의 유일한 해쉬함수(h(x))를 가진다. 이 해쉬함수는 자신의 대표 id 를 만들기 위해서 사용되고 파일 저장, 검색시에도 사용된다. 일반적인 P2P 파일 저장 방법은 그림 6 와 같다. 파일을 저장하기 위해서는 피어A는 저장할 파일(a)의 특성(파일 이름, 크기, 생성날짜, 그외 정보) 를 해쉬함수(h(x))를 이용해서 저장위치키(destination key) 를 결정한다. 이 저장위치키를 이용해서 이 값을 담당하는 피어 Z 를 찾아낸다. 그리고 파일(a) 를 피어 Z 에 저장한다. 피어B 에서 해당 파일(a)를 찾기 위해서는 해쉬함수(h(x))를 이용해 알아낸 저장위치키를 이용해서 피어Z를 찾아서 파일(a)를 얻는다. 이 경우에 객체(a) 의 정보를 정확하게 알아야만 찾을 수 있고 피어 Z 가 P2P 네트워크에서 탈퇴하면 저장했던 파일(a) 는 함께 접근 불가능해진다. 이러한 문제를 해결하기 위해서 가용성 향상 기법이 필요하다.

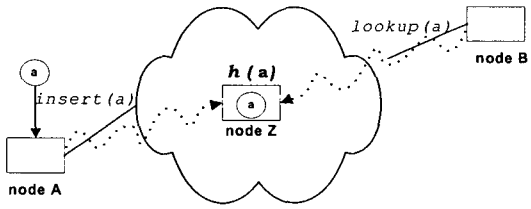



그림 6. P2P시스템에서의 파일 저장 방법
Fig 6. File storing method in P2P system

### 4.2 복제 기반 고가용성 파일 저장 방법

4.2.1 복제방법(Replication)

복제 방법은 저장할 파일을 P2P 네트워크의 여러 피어에 복제해서 중복 저장한다. 복제하는 방법으로 다중 해쉬 방식과 재귀 해쉬 방식이 있다. 다중 해쉬 방식은 복제할 수(R개)만큼 해쉬 함수(h1(x), h2(x), h3(x), … , hR(x))를 만들어 복제를 저장할 때 사용한다. 재귀 해쉬 방식은 해쉬함수의 결과값을 다시 해쉬한다. 맨 처음 파일 저장시의 해쉬 함수는 h(x) 이고 두번째 복제 파일 저장시의 해쉬 함수는 h(h(x)) 가 된다. 두 방식은 성능면에서 차이는 없다. 본 논문에서 다중 해쉬 방식을 채택하였다. 다만, 복제된 파일이 같은 피어에 저장되는 것은 금지한다. 복제 방식은 복제하는 횟수만큼 가용성이 향상되는 것을 보장한다.

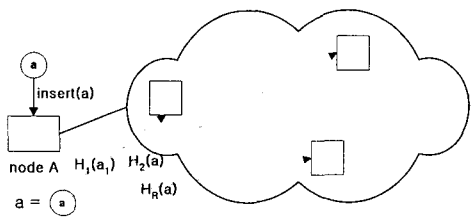



그림 7. 복제 방법
Fig 7. Replication scheme

4.2.2 분할 방법(Fragmentation)

분할 방법은 저장할 파일(a)을 다수(F)의 조각(fi)으로 분할해서 각각을 P2P 네트워크 저장하는 방식이다. 이 방법은 파일을 분할하지 않고 저장할 경우 해당 피어가 탈퇴하면 전체 파일을 모두 잃어버릴 수 있기 때문에 부분적이라도 파일 사용이 필요할 때 적합하다. 그리고 모든 파일 조각이 접근 가능할 경우는 병렬 수집이 가능한 장점이 있다.

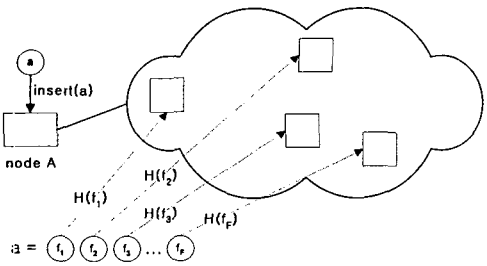



그림 8. 분할 방법
Fig 8. Fragmentation scheme

4.2.3 복제 분할 방법(Replicated Fragmentation)

이 방법은 위의 복제 방법과 분할 방법을 혼합한 것으로 파일을 다수(F개)의 조각(fi)으로 분할한 뒤 각 조각을 다수(R개)로 복제해서 각각을 저장하는 방식이다. 그림 9 에서 복제 분할 방법을 설명하고 있다. 파일(a)는 R개 복제되고 각 복제파일(ai)는 F개의 조각(f1, f2, f3, ⋯ , fF)으로 분할되었다. 이 각각의 조각을 해쉬함수 Hi(x)를 이용해서 저장할 피어를 정하도록 하였다.

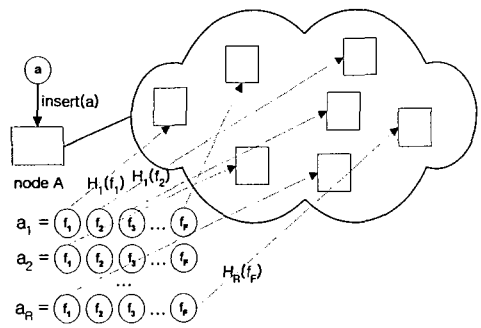



그림 9. 복제분할 방법
Fig 9. Replicated fragmentation scheme

복제 분할 방법은 저장된 객체를 다시 얻어내기 위해서는 가장 효과적이지만 객체를 완성 조립하기 위해서 많은 수의 메시징이 필요하고 또한 저장공간의 효율이 떨어질 수 있다. 또한 복제 분할 방법을 선택했다 하더라도 기본 MagicSquare 시스템의 저장 방식으로는 가장 신뢰도가 낮은 피어에 객체 저장이 될 가능성이 높으므로 저장된 객체의 가용성을 높이기 힘든다. 따라서, 어느 정도 신뢰도가 보장이 되는 피어에 객체를 저장해야 한다. 이를 위해서 피어의 라우팅 테이블 크기가 어느 이상되는 피어에만 객체를 저장하는 방법을 사용해야만 저장된 객체의 가용성을 높일 수 있다.

## V. 성능평가 및 고찰

이 장에서는 시뮬레이션을 통해서 본 논문에서 제안하는 MagicSquare 프로토콜의 성능을 평가한다. 본 논문에서 서술한 의사 코드를, Java 를 사용해서 메시지 수준의 시뮬레이터를 작성하였다.

### 5.1 실험환경

MagicSquare 의 성능을 알아보기 위해서 다음과 같은 실험 조건을 가정하였다.

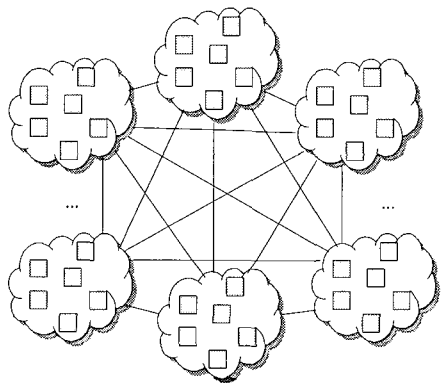


그림 10. 시뮬레이션 네트워크 토폴로지
Fig 10. Simulation network topology

그림 10 은 실험에 사용된 네트워크 토폴로지를 나타낸다. 전체 시스템은 다수의 로컬 네트워크로 존재하고 각 로컬 네트워크에는 다수의 피어 집단이 존재한다. 로컬 네트워크 간에는 네트워크 토폴로지가 존재하지 않고 실험 초기에 임의적으로 메시지 전송시간을 양방향으로 설정한다.

표 1 에서 메시지 전송시간 범위 및 로컬 네트워크에 존재하는 피어의 수를 정하였다. 각 피어의 성능을 구분하기 위해서 피어의 메시지 재전송 처리 시간을 실험 초기에 표 1 의 값으로 임의로 가정하였다. 피어 결함 정도를 정하기 위하여 피어의 참여 신뢰도를 가정하여 사용하였다. 각 피어는 0등에서 99등까지 100단위로 나누어서 Zipf 분포로 시스템에 참여할 가능성을 설정하여 실

험에서 설정한 오류율 이하일 경우에는 해당 피어에 결함이 발생했다고 가정했다. 예를 들어 오류율이 5% 인 경우에는 피어의 결함 비율이 5% 이하인 피어에 결함이 발생했다고 가정했다.

표 1. 실험 파라메터
Table 1. Simulation parameters

| 중요 실험 인자 | 값 |
|---|---|
| 시스템에 참가한 피어의 수 (N) | 100 ~ 100,000 |
| 로컬 네트워크 수 (M) | 1 ~ 1000 |
| 로컬 네트워크에속한 피어의 수 ($L_M$) | N/M |
| 로컬 네트워크간 메시지의 외부 전송 지연 시간($D_G$) | 10ms ~ 1000ms |
| 로컬 네트워크 내부 메시지 전송 지연시간($D_L$) | 1ms |
| 피어($P_i$)의 메시지 재전송 처리 시간 ($P_{delay}(i)$) | 1ms ~ 100ms |
| 피어의 결함 비율 ($P_{fault}(i)$) | $\frac{1}{random(100)}$ |
| 피어의 성능(P(i)) | $\frac{1000}{P_{delay}(i)} \times P_{fault}(i)^2$ |

### 5.2 탐색

MagicSquare 의 성능은 질의에 대한 응답을 처리할 때 방문하는 피어 수에 의존하는데, MagicSqure 의 탐색 방법은 N개의 피어를 가지는 스킵리스트의 탐색 방법과 비슷하기 때문에 O(logN)에 근접할 것이다. 시뮬레이션에서 선택된 각 피어는 시스템에서 랜덤 키를 찾는 질의를 요청하고, 질의를 처리할 때 접속하는 피어의 수를 측정하여서 탐색의 성능을 알아보았다.

그림 11 는 시스템에 참가하는 피어의 수가 늘어남에 따라 시스템의 성능, 즉 메시지의 홉 수의 변화를 보여준다. 피어의 수와 홉 수의 그래프가 logN 그래프에 근접하고 있음을 볼 수 있다. 따라서 시스템에서 데이터 탐색 비용은 O(logN)에 근접한다. MagicSquare 시스템에서는 각 피어의 최고 라우팅 크기가 메시지 전송 홉스에 영향을 미친다. 실험 결과에서 살펴보면 최고 라우팅 크기를 4로 설정한 MS(Lv4) 와 8로 설정한 MS(Lv8) 의 성능이 전체 시스템의 규모가 커짐에 따라 역전되는 것을 알 수 있다. MagicSquare 시스템의 규모가 작을 경우에는 각 피어의 최고 라우팅 크기를 적절한 선에서 제한하는 것이 필요하다고 해석되며 시스템의 규모를 키울 경우에는 최고 라우팅 크기를 키우는 것이 바람직해 보인다. 또한, 실험 결과에서 앵커 피어로의 추가 링크도 메시지 전송 홉수에 영향을 미침을 알 수 있다.

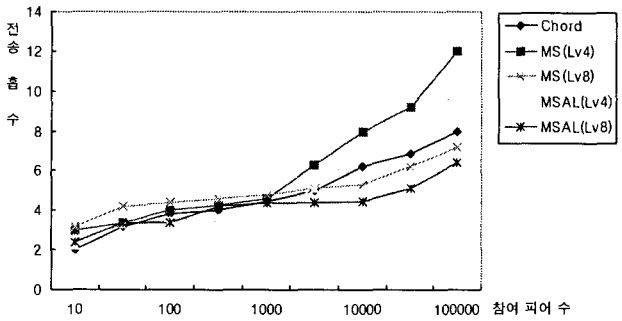



그림 11. 시스템에 참여하는 피어 수의 변화에 따른 메시지 전송 평균 홉 수
Fig 11. Average hop counts of message routing with the number of peers

### 5.3 가용성 (Availability)

앞에서 소개한 복제를 통한 가용성 향상 기법의 효과를 비교해 본다.

분석에 필요한 파라메터는 다음과 같다.

- 네트워크에 참여하는 피어 수 : N
- 결함(Failure)있는 피어 수 : M
- 파일 조각의 수 : F
- 복제 파일의 수 : R

파일 복제나 파일 분할을 사용하지 않았을 경우(No replication, no fragmentation) 는 시스템에 저장된 파일이 가용하지 않을 확률은 파일을 저장하고 있는 피어가 결함이 될 확률이다.

$$Failure(a) = \frac{M}{N}$$

파일 복제만 적용한 경우(No fragmentation, replication) 에는 복제 파일이 저장된 모든 피어가 결함이 될 확률이다. 단, 복제 파일은 같은 피어에 저장될 수 없으므로 복제 파일은 모두 다른 피어에 저장되었을 경우를 고려해야 한다.

$$Failure(a) = \frac{M}{N} \times \frac{M-1}{N-1} \times \cdots \frac{M-R+1}{N-R+1}$$

파일 분할만 적용한 경우(Fragmentation, no replication)에는 모든 조각 파일이 접근가능해야 한다. 조각 파일이 저장된 단 하나의 피어가 결함이 있다면 전체적으로 파일 재구성에 결함이 발생한다. 조각 파일은 같은 피어에 저장될 수 있다. 이를 고려해서 파일 재구성에 실패할 확률은 다음과 같다.

$$Failure(a) = 1 - \left(\frac{N-M}{N}\right)^F$$

파일 분할과 파일 복제가 함께 적용한 경우(Fragmentation, replication)에는 조각파일의 가용 확률을 먼저 계산해야 한다. 조각 파일의 가용 확률은 조각파일이 저장된 모든 피어가 결함이 될 확률로부터 구할 수 있다.

$$Available(a_i) = 1 - \frac{M}{N} \times \frac{M-1}{N-1} \times \cdots \frac{M-R+1}{N-R+1}$$

이제 전체 파일의 가용 확률은 개별 조각 파일의 가용 확률의 곱이다.

$$Available(a) = \left(1 - \frac{M}{N} \times \frac{M-1}{N-1} \times \cdots \frac{M-R+1}{N-R+1}\right)^F$$

따라서, 파일 분할과 파일 복제가 함께 적용된 경우의 파일이 가용하지 않을 확률은 다음과 같다.

$$Failure(a) = 1 - \left(1 - \frac{M}{N} \times \frac{M-1}{N-1} \times \cdots \frac{M-R+1}{N-R+1}\right)^F$$

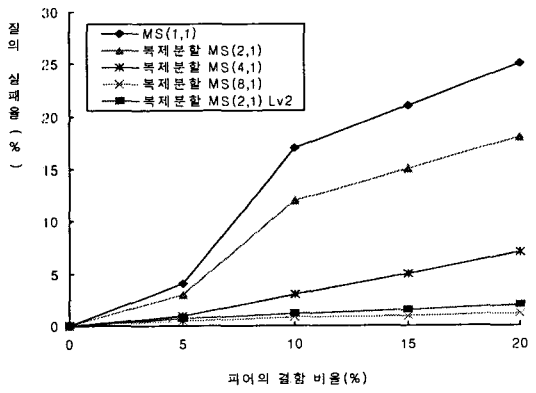



그림 12. 객체 복제시 피어 결함 비율에 대한 질의 실패율

Fig 12. Average query failure rate with peer failure rate in object replication scheme

그림 12 는 복제 분할 기법의 효과를 측정하기 위해서 피어의 결함 비율에 따른 메시지 전송 평균 홉 수를 측정한 결과이다. 이 실험에는 1000개의 피어 (N=1000) 가 참여하고, 최대 라우팅 테이블의 높이를 4로 설정하였다. 전체 피어의 수는 본 실험에서는 질의한 객체가 재구성할 수 없는 경우를 질의 실패율로 측정하였다.

그림 12 의 실험에서 복제 기법의 효용성을 검증할 수 있었다. 객체 복제를 하지 않은 기본 MagicSquare 시스템은 피어의 결함 비율이 높아짐에 비례하여 질의실패율이 증가되었으나 복제 횟수를 늘임에 따라 획기적으로 질의실패율이 감소함을 알 수 있다. 또한, 객체를 저장함에 있어 라우팅 레벨을 어느 이상으로 제한한 경우가 복제 횟수를 늘이는 것 보다 더욱더 효과적임을 알 수 있다. 8번 복제한 실험 결과와 2번 복제하고 저장 피어의 레벨을 2 이상으로 제한한 경우의 질의 실패율이 거의 유사함을 알 수 있다.

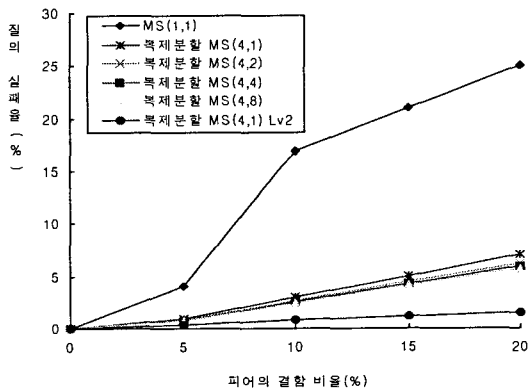



그림 13. 객체 복제분할시 피어 결함 비율에 대한 질의 실패율

Fig 13. Average query failure rate with peer failure rate in object replicated fragmentation scheme

그림 13 의 실험에서 분할 기법의 효과를 측정하였다. 앞서 측정한 결과와 대비를 위해서 복제를 4번한 경우를 실험하였다. 실험 결과로는 분할 기법은 질의 실패율을 거의 줄이지 못하였다. 이 결과를 보면 분할 기법 보다는 복제 기법이 객체 질의 성공률을 높일 수 있음을 보여준다. 다만, 분할 기법은 객체가 전체 질의가 아니라 부분 질의가 효과적인 경우에 질의 성공률에 영향을 미칠 수 있다고 예측된다.

또한, 이 실험에서도 객체를 저장함에 있어 라우팅 레벨을 어느 이상으로 제한한 경우가 복제 횟수를 늘이는 것 보다 더욱더 효과적임을 알 수 있다.

## VI. 결 론

MagicSquare 는 스킵리스트의 사상을 이용하여 랜덤 속성을 가진 분산 해쉬 테이블 방식의 P2P 네트워크 기술이다. MagicSquare 는 이미 개발된 Chord나 Pastry와 같은 분산 해쉬 테이블 P2P 시스템의 라우팅 테이블을 랜덤하게 구성하여 피어의 구성이 최악의 경우에도 라우팅 테이블을 구성하는 비용과 키를 탐색하는 비용이 평균 탐색 비용인 O(logN) 에 근접한다. 또한, P2P 시스템의 가용성은 매우 중요한 요소이다. 다양한 가능성을 보이는 P2P 시스템의 적용분야가 파일 공유 등 일부분에 국한되는 이유도 가용성 제공 보장이 어려워 시스템의 신뢰성을 제공하기 어렵기 때문이다. MagicSquare 에서는 시스템에 저장되는 객체를 중복저장하여 가용성을 높이고자 하였다. 객체의 중복 저장방식을 P2P 시스템의 특징에 맞춰 복제 방법, 분할 방법, 복제 분할 방법을 제시하여 가장 타당한 방법을 시뮬레이션을 통해서 검증하였다. 앞으로 동적인 인터넷 환경을 고려하여 대규모 피어의 접속 및 네트워크 분할 등의 상황을 고려하는 방법에 대한 연구가 필요할 것으로 보인다.

## 참고문헌


[1] A. Oram, "Peer-to-Peer", O'Reilly, Mar, 2001.

[2] ThinkStream, "A Technical Review of the Next Generation Internet Architecture",

[3] C. Shirky, "What is P2P... and what Isn't", http://www.openp2p.com/pub/a/472

[4] William Pugh, "Skip Lists: A Probabilistic Alternative to Balanced Trees", Communications of the ACM, Vol. 33, Jun 1990

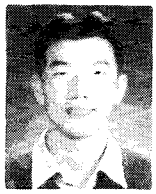
[5] I. Stoica, R. Morris, D.Liben-Nowell, D. R. Karger, M. Frans Kaashoek, F. Dabek, H. Balakrshinan, "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications", IEEE/ACM Transactions on Networking, Vol. 11, Feb 2003.

[6] A. Rowstron, P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems", Proceedings of the 18th IFIP/ACM Int'l Conf. on distributed Systems Platforms, Nov. 2001


## 저자소개


**손 영 성(Young-Sung Son)**

995 부산대학교 전자계산학과 (이학사)
997 부산대학교 전자계산학과 (이학석사)
2006 부산대학교 컴퓨터공학과(공학박사)
1997-현재 한국전자통신연구원 선임연구원
※관심분야 : 홈네트워크, P2P네트워크, 상황인지컴퓨팅



**정 일 동(Il-dong Jung)**

2000 부산대학교 전자계산학과 (이학사)
2002 부산대학교 전자계산학과 (이학석사)
2002~현재 부산대학교 컴퓨터공학과 (박사과정)
2003-현재 LG전자 DAC연구소 주임연구원
※관심분야 : 정보가전, 인터넷컴퓨팅, 임베디드시스템

**김 경 석(Kyongsok Kim)**

1977 서울대학교 무역학과 (경제학사)
1979 서울대학교 전자계산학과 (이학석사)
1988 일리노이 주립대 (어바냐-샴페인) (전자계산학 박사)
1988-1992 미국 노스다코다 주립대학교 전자계산학과 조교수
1992-현재 부산대학교 전자전기정보컴퓨터공학부 교수
※관심분야 : 데이터베이스, 멀티미디어, 한글/한말 정보처리, 인터넷컴퓨팅