하이퍼큐브 멀티컴퓨터를 위한 분산 상호배제 알고리즘의 설계 및 평가

하 숙 정[†] · 배 인 한[††]

요 약

공유자원의 상호배제를 위한 분산 상호배제 알고리즘은 두 가지 방식으로 접근되어 왔으며 토큰-기반과 인가-기반의 두 부류로 나누어 진다. 토큰-기반 알고리즘은 노드들 간에 단 하나의 토큰을 공유하며 이 토큰을 가지고 있는 노드만이 공유자원에 접근할 수 있도록 한다. 인가-기반 알고리즘은 공유자원 접근에 대한 인가를 받기 위해서 여러 노드들 간에 1회 이상의 메세지 교환이 필요하다. 지난 몇 년 동안 하이퍼큐브 구조는 단순하면서도 풍부한 위상을 가지고 있어서 멀티프로세서 시스템에서 많이 수용되어 왔다. 따라서 본 논문에서는 하이퍼큐브를 위한 분산 상호배제 알고리즘을 연구하여 하이퍼큐브에 적합한 새로운 정보구조에 기초한 분산 상호배제 알고리즘을 설계한다. 새로운 정보구조는 하이퍼큐브에 삽입된 논리적 메쉬로부터 얻어낸 T-패턴의 요청집합이다. 임의의 노드가 공유자원에 접근하고자 한다면, 그 노드는 요청집합 내의 모든 노드들에게 요청 메세지를 전송하고, 요청집합 내의 모든 노드들로부터 인가 메세지를 받은 후에 공유자원에 접근한다. 본 논문에서 제안하는 알고리즘의 성능을 최소 왕복 지연, 블럭킹 지연, 그리고 공유자원 접근당 메세지 개수로 평가한다.

Design and Evaluation of a Distributed Mutual Exclusion Algorithmfor Hypercube Multicomputers

Sook-Jeong Ha[†] · Ihn-Han Bae[††]

ABSTRACT

Distributed mutual exclusion algorithms have employed two approaches to achieve mutual exclusion and can be divided into two broad classes:token-based and permission-based. Token-based algorithms share a unique token among the nodes and a node is allowed to access its common resources if it possesses the token. Permission-based algorithms require one or more successive rounds of message exchanges among the nodes to obtain the permission to access the common resources. A hypercube architecture has earned wide acceptance in multiprocessor systems in the past few years because of its simple, yet rich topology. Accordingly, we study distributed permission-based mutual exclusion algorithms for hypercubes, and design a distributed permission-based mutual exclusion algorithm based on a new information structure adapted to the hypercubes. The new information structure is a request set of T-pattern from a logical mesh that is embedded into a hypercube. If a node wants to access the common resources, it sends request message to all nodes in the request set by Lan's multicast algorithm. Once the node receives a grant message from all nodes in the request set, it accesses the common resource. We evaluate our algorithm with respect to minimum round-trip delay, blocking delay, and the number of messages per access to the commom resource.

† 정 회 원:대구효성가톨릭대학교 대학원 전산통계학과
†† 정 회 원:대구효성가톨릭대학교 전자정보공학부 부교수
논문접수:1997년 4월 30일, 심사완료:1997년 8월 8일

1. 서 론

멀티 노드 시스템에서의 노드들을 효과적으로 상호 연결하기 위한 많은 위상들이 지난 몇 년 동안 제안되어 왔다. 특히 하이퍼큐브 구조는 단순하면서도 풍부한 위상을 가지고 있어서 SIMD와 MIMD 구조에 널리 사용되어 왔다. MIMD 기계 상의 알고리즘의 성능은 동기화 문제를 해결하기 위해 사용되는 방법에 많은 영향을 받는다. 동기화 문제 중의 대표적인 것으로 상호배제 문제가 있다. 상호배제는 운영체제의 근본적인 문제로서 공유자원에 접근하고자 하는 서로 독립된 여러 프로세스 중에 단 한 개의 프로세스만이 공유자원에 접근하도록 보장해주는 문제이다. 집중형 시스템에서는 이 문제에 대한 많은 해결책이 제안되었으나 분산 시스템에서는 몇몇 알고리즘만이 제안되어 왔다[10].

분산 상호배제 알고리즘은 인가-기반(permission-based) 알고리즘과 토큰-기반(token-based) 알고리즘 두 가지 그룹으로 분류될 수 있다[14]. 인가-기반 알고리즘에서 공유자원에 접근하고자 하는 노드는 특정 노드 집합으로부터 인가를 받은 후에만 접근할 수 있다. Ricart와 Agrawala[12]는 Lamport가 제안한 알고리즘[7]에 연기된 인가(defered grants)를 사용하여 알고리즘의 성능을 개선하였는데, 공유자원에 접근하려는 노드는 모든 다른 노드로부터 인가를 받아야만 접근할 수 있다. Maekawa[9]는 Ricart와 Agrawala의 알고리즘에서 상호배제를 하기 위해 필요한 메세지의 개수를 최적화시켰다. Gupta[5]는 하이퍼큐브 구조를 완전히 이용하는 선택적 방송(selective broadcast)이라는 분산 대칭 상호배제 알고리즘을 제안하였다.

토큰-기반 알고리즘에서는 모든 노드들이 단 한 개의 토큰을 공유하며, 이 토큰을 가지고 있는 노드만이 공유자원에 접근할 수 있다. 토큰이 유일하게 존재하므로 분산 시스템에서의 상호배제를 확신할 수 있다. Suzuki와 Kasami[15]의 알고리즘에서는 토큰을 가지고 있지 않은 노드가 공유자원에 접근하고자 한다면 다른 모든 노드에게 토큰 요청 메세지를 방송한다. 토큰을 가지고 있는 노드가 요청 메세지를 받게 되면, 토큰을 요청노드에게 전송한다. Naimi와 Trehel[16]은 요청노드에게 루트노드로의 경로를 제공하기 위해 자신에게 도착한 가장 최근의 노드만을 저장하고 있는 마지막(last) 포인터를 유지하며, 요청노드가 토큰을 받아 공유자원을 사용한 후 토큰을 넘겨 줄 자신 바로 다음의 요청노드만을 기록하고 있는 분산 큐 자료구조를 가지고 역(reversal) 경로를 사용하는 알고리즘을 제안하였다.

토큰-기반 알고리즘에서는 하나의 토큰이 시스템에 존재하므로써 인가-기반 알고리즘에서 필요한 여러 노드로부터의 개별적 응답 메세지 대신에 필요한 노드들 모두의 응답 메세지를 대변하는 것으로 볼 수 있는 한 개의 토큰만이 전송되기 때문에 상호배제에 사용되는 메세지 개수가 일반적으로 적다. 그러나 요청 메세지가 토큰이 존재하는 노드까지 순차적으로 따라가야 하므로써 걸리는 지연 시간이 일반적으로 길다. 또한 노드에 결함이 생긴 경우 이를 해결하는 알고리즘이 복잡해진다[16]. 인가-기반 그룹에 대해서는 이들을 통합하기 위한 일반 이론이 존재하지만 토큰-기반 알고리즘들에 대해서는 이러한 일반적인 이론이 없다[17]. 이 두 그룹간에는 이러한 메세지 복잡도와 속도간의 상반관계가 존재한다.

본 논문에서는 하이퍼큐브 멀티컴퓨터 상에서 상호배제를 효과적으로 해결하기 위한 인가-기반 알고리즘을 설계한다. 이 알고리즘은 하이퍼큐브 구조에 삽입된 논리적 메쉬 구조를 기반으로 T-패턴의 정보 구조를 요청집합으로 사용한다. 제안하는 알고리즘의 성능을 최소 왕복 지연, 블럭킹 지연, 그리고 공유자원 접근당 메세지의 개수로서 평가한다. 본 논문의 구성은 다음과 같다. 2장에서는 하이퍼큐브 멀티컴퓨터에 관하여 설명하고, 3장에서는 하이퍼큐브를 위한 상호배제에 관한 관련 연구들을 살펴본다. 4장에서는 본 논문에서 제안하는 하이퍼큐브를 위한 인가-기반 분산 상호배제 알고리즘을 설계하고, 5장에서는 제안하는 알고리즘의 성능을 n-차원의 하이퍼큐브에서의 최소 왕복 지연, 블럭킹 지연, 그리고 공유자원 접근당 메세지 개수로 평가한다. 그리고 마지막으로 6장에서 결론을 맺는다.

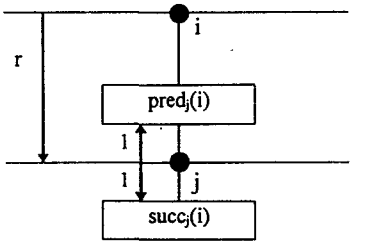
2. 하이퍼큐브 멀티컴퓨터

n-차원 하이퍼큐브는 2^n개의 노드로 구성되며 재귀적으로 정의될 수 있다[6]. 0-차원 하이퍼큐브는 하나의 노드로 구성되며, 1-차원 하이퍼큐브는 두 개의 0-

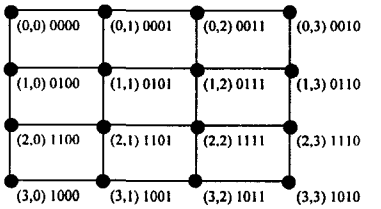
차원 하이퍼큐브를 연결하므로써 만들어진다. 즉 일반적으로 (n+1)-차원 하이퍼큐브는 두 개의 n-차원 하이퍼큐브를 서로 대응하는 노드끼리 연결하므로써 만들어진다. 하이퍼큐브의 각 노드에는 각각을 구분할 수 있도록 차원 수 만큼의 비트로 표현되는 레이블이 붙게 된다. 두 개의 n-차원 하이퍼큐브를 연결할 때, 임의의 한 하이퍼큐브의 노드 레이블에는 0이 앞에 붙게 되며 다른 하이퍼큐브의 노드 레이블에는 1이 앞에 붙는다. 즉 n-차원 하이퍼큐브 Q_n의 각 노드는 d_0에서 d_{n-1}까지의 차원값 (d_{n-1}, d_{n-2}, ..., d_2, d_1, d_0)에 해당하는 링크를 따라 다른 노드와 연결되어 있으며 각 레이블은 비트열 ($a_{n-1}a_{n-2}...a_2a_1a_0$)으로 표현될 수 있다.

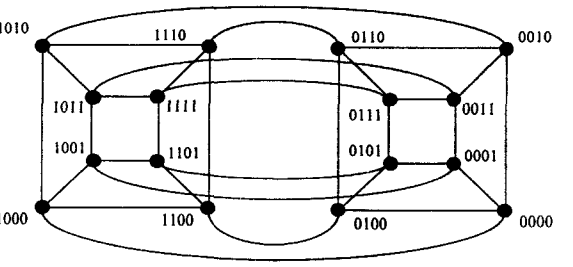
하이퍼큐브 멀티컴퓨터는 다음과 같은 성질을 갖는다[6].

- 두 노드는 직접적인 링크에 의해 연결될 수 있는데, 이 때 두 노드의 레이블은 단 한 개의 위치에서만 비트값이 다르다.
- 하이퍼큐브에서의 두 노드 x와 y의 레이블을 l_x와 l_y라고 하자. l_x와 l_y를 구성하는 각 비트에 대해 서로 값이 다른 비트 위치의 개수를 두 노드간의 해밍 거리(Hamming distance)라 한다. 예를 들어 레이블 101과 010의 해밍 거리는 3이다. 해밍 거리는 ⊕가 비트간의 exclusive-or 연산일 때, $l_x \oplus l_y$를 수행한 결과 값에서 1의 개수가 된다. 두 노드간의 최단 경로에 포함되는 통신 링크의 수는 두 레이블간의 해밍 거리이다. 노드 x와 y간에 전송되는 메세지는 $l_x \oplus l_y$에서 비트 값이 1인 위치에 해당하는 차원을 따라 전송되므로써 라우트될 수 있다. n-차원 하이퍼큐브의 노드 레이블에는 n개의 비트가 있으므로 두 노드간의 최단 경로에는 n개 이하의 링크가 포함된다.
- n-차원의 하이퍼큐브 Q_n의 각 노드 j는 $i \in \{1, 2, 3, ..., 2^n\}$일 때, $pred_j(i)$와 $succ_j(i)$라는 2×2^n개의 집합을 가진다. 노드 i와 j간의 거리를 r이라 하자. $pred_j(i)$는 j의 이웃 중에서 i로의 거리가 1작은 즉, i로부터의 거리가 r-1인 이웃 노드의 집합이며, $succ_j(i)$는 i로부터의 거리가 1 더 많은 즉, i에서의 거리가 r+1인 이웃 노드의 집합이다. $pred_j(i)$와 $succ_j(i)$의 cardinality인 $|pred_j(i)| = r$이고 $|succ_j(i)| = n - r$이다. 모든 i에 대해, $pred_i(i) = \phi$이며 $succ_i(i)$는 노드 i의 모든 이웃 n개를 포함한다. 다시 말해, $pred_j(i)$는 i와 j의 레이블간에 값이 다른 비트 위치와, $succ_j(i)$는 레이블간에 같은 값을 갖는 비트 위치와 관련된 차원을 따라 j에 연결되어 있는 이웃 집합으로 정의될 수 있다. (그림 1)은 노드 j에 대한 $pred_j(i)$와 $succ_j(i)$를 보여준다.



(그림 1) $pred_j(i)$와 $succ_j(i)$
(Fig. 1) $pred_j(i)$ and $succ_j(i)$





(그림 2) 4×4 메쉬를 4-차원 하이퍼큐브로의 매핑
(Fig. 2) Mapping a 4×4 mesh to processors in a four-dimensional hypercube

- $2^r \times 2^s$ 순환(wraparound) 메쉬는 2^{r+s} 개의 노드를 갖는 (r+s)-차원 하이퍼큐브로 삽입될 수 있다. G(i, r)은 r개의 비트로 표현된 i의 그레이(Gray) 코드이며, |는 두 그레이 코드 간의 연결(concatenation)을 나타낸다고 하자. 메쉬의 주소가 (i, j)인 노드는 하이퍼큐브에서 레이블이 G(i, r)|G(j, s)인 노드로 매핑된다. 메쉬에서 특정 노드의 바로 옆 이웃 노드들은 하이퍼큐브에서 대응하는 노드와의 레이블 간의 비트값이 한 곳에서만 다른 노드로 매핑된다. (그림 2)는 $2^2 \times 2^2$ 메쉬가 2^4개 노드를 가지는 하이퍼큐브에 삽입된 것을 보인다. 이 때 r과 s는 2이다. 그러므로, 메쉬의 노드 (i, j)는 하이퍼큐브의 노드 G(i, 2)|G(j, 2)에 매핑된다.

메쉬를 하이퍼큐브에 매핑하므로써 메쉬에서 같은 행에 있는 모든 노드들은 하이퍼큐브에서 레이블내 r개의 최상위 비트들이 동일한 노드에 매핑되며, 같은 열에 있는 모든 노드들은 s개의 최하위 비트들이 동일한 레이블에 해당하는 노드로 매핑된다.

3. 관련연구

3.1 Gupta의 연구

분산 상호배제를 위한 Sanders의 알고리즘[13]에서, 각 노드 i는 정보구조로서 2개의 관련 집합, 요청집합 R_i와 통지 집합 I_i를 가진다. 임의의 노드 i가 상호 배제적으로 자원에 접근하기 위해서는, 먼저 요청집합 R_i에 있는 모든 노드로부터 인가를 받아야 하며, 자원을 해제할 때는, 통지집합 I_i에 있는 모든 노드들에게 통지해야 한다. 요청에 대한 인가를 부여하는 동안에 발생하는 충돌을 해결하기 위해서 요청들은 타임 스탬프 값에 따라서 순서화된다. Sanders의 알고리즘에서 각 노드는 공유자원에 접근하고 해제하기 위해 다음과 같은 프로토콜을 실행한다:

- REQUEST: R_i에 있는 모든 노드에게 타임 스탬프가 붙은 REQUEST 메세지를 보내고, R_i내 모든 노드로부터 GRANT 메세지를 기다린다.
- ACCESS: 자원에 접근한다.
- RELEASE: I_i내 모든 노드에게 RELEASE 메세지를 보낸다.

Gupta의 알고리즘에서는 통지집합 I_i와 요청집합 R_i가 동일하며 S_i로 나타낸다. Gupta의 알고리즘에서 기본 개념은 집합 S_i에 요청노드 i로부터 $\lceil n/2 \rceil$ 거리 안에 있는 노드만을 포함하게 하는 것이다. 만일 임의의 두 노드 i와 j가 공유자원에 접근하고자 할 때, i와 j간의 최대 거리는 n이므로 S_i와 S_j간에는 적어도 한 개의 공통된 노드가 존재하며, REQUEST 메세지를 받는 노드는 어느 한 순간에 단 하나의 노드에게만 자원에 접근할 수 있는 인가를 줄 수 있으므로, 공통 노드는 조정자(arbitrator)와 같이 행동하여 노드 i와 j 둘 다에게 GRANT 메세지를 전송하는 일은 없다.

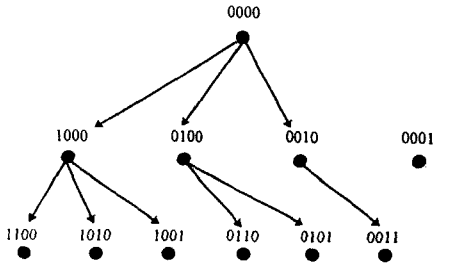
Gupta의 상호배제 알고리즘은 다음과 같이 세 부분으로 구성된다.

- 첫번째 부분은 공유자원에 접근하고자 하는 요청노드인, 시작노드 i가 하는 일이다. 노드 i는 $m = \lceil n/2 \rceil$ 일 때 차원 $d_n, d_{n-1}, \ldots, d_m$을 따라 도착할 수 있는 $succ_i(i)$ 노드들에게만 REQUEST 메세지를 멀티캐스트한다.
- 두번째 부분은 요청집합내의 중간(intermediate) 노드가 하는 일이다. 차원 d_{min}을 따라 REQUEST 메세지를 받은 노드 i로부터 거리 $r(1 \le r < m)$에 있는 중간 노드는 차원 $d_{min-1}, d_{min-2}, \ldots, d_{m-r}$을 따라 도착할 수 있는 $succ_j(i)$의 모든 노드에게 그 메세지를 전송한다.
- 세번째 부분은 알고리즘이 끝나는 시기에 해당한다. 메세지를 전송하는 것은 시작노드 i로부터 거리 m에 있는 노드에서 끝난다.

S_i는 노드 i로부터 $\lceil n/2 \rceil$ 거리 내에 있는 모든 노드들의 집합이다. 그러나 Gupta의 선택적 방송에서는 S_i의 모든 노드를 방문하지 않는다. S_i^*는 선택적 방송에서 실제로 방문하는 노드의 집합을 나타낸다고 하자. 이와 비슷하게 후속자의 선행자 집합에도 같은 의미로 위첨자 *를 붙인다. 선택적 방송은 노드 i로부터 S_i^*내의 임의의 수신노드 j로 경유해 가는 유일한 경로를 제공해 준다. 노드 j로 가는 REQUEST 메세지는 i와 j의 레이블 간에 서로 다른 값을 갖는 비트위치에 연관된 차원을 따라서만 경유되며, 가장 높은 차원에서 시작하여 차례로 더 낮은 차원을 따라 경유한다. 노드 i로부터 $\lceil n/2 \rceil$ 거리에 있는 노드 k가 일단

REQUEST 메세지를 받으면, 이 노드는 노드 i에게 자원에 대해 인가해 줄 수 있는지를 결정한다. 만일 인가할 수 있다면, 노드 k는 $pred_k^*(i)$ 노드에게 GRANT 메세지를 전송한다.

임의의 중간 노드 k는 $succ_k^*(i)$내의 모든 노드로부터 GRANT 메세지를 받을 때까지 기다렸다가 노드 i에게 자원을 인가해 줄 수 있게 되면, 노드 k의 $pred_k^*(i)$ 노드에게 GRANT 메세지를 전송할 것이다. 이러한 역전송은 S_i^*내의 모든 중간 노드에서 수행된다. 이러한 방식으로, GRANT 메세지는 REQUEST 메세지가 전송된 경로를 역으로 따라간다. 일단 노드 i가 $succ_i^*(i)$ 집합에 있는 모든 노드로부터 GRANT 메세지를 받으면, 자신의 S_i^*에 있는 모든 노드로부터 필요한 인가를 얻었다는 것을 알고, 공유자원의 접근을 진행한다. 노드 i가 더 이상 그 자원이 필요하지 않게 되면, S_i^*내의 모든 노드에게 RELEASE 메세지를 방송한다. RELEASE 메세지는 REQUEST 메세지와 같은 경로를 따라간다. (그림 3)은 4차원 하이퍼큐브 Q_4에서 노드 0000가 공유자원에 접근하기 위해 선택적 방송에서 생성되는 REQUEST 메세지를 보여 준다.



(그림 3) 선택적 방송에서 생성된 REQUEST 메세지의 예
(Fig. 3) REQUEST messages generated in the selective broadcast example

n-차원 하이퍼큐브 Q_n에서 공유자원 접근 요청 노드 i와 S_i^*내의 가장 멀리 떨어진 노드 간의 거리는 $\lceil n/2 \rceil$이고, S_i^*의 cardinality $|S_i^*| = \binom{n+1}{\lceil n/2 \rceil}$이다. 그리고 요청노드 i로부터 거리 d에 있는 S_i^*내의 노드의 개수는 $\binom{\lceil n/2 \rceil + d}{d}$이다. 따라서 선택적 방송을 사용하는 Gupta의 알고리즘의 성능은 다음과 같다.

• 최소 왕복 지연:

$$D = 2 \times \left\lceil \frac{n}{2} \right\rceil$$

• 블럭킹 지연:

$$B = \frac{1}{N-1}\left[\left\{\sum_{d=1}^{\lceil \frac{n}{2} \rceil} d \binom{\lfloor \frac{n}{2} \rfloor + d}{d}\right\} + 2\left\lceil \frac{n}{2} \right\rceil \left\{N - \binom{n+1}{\lceil \frac{n}{2} \rceil + d}\right\}\right]$$

• 공유자원 접근당 메세지 개수:

$$\text{Min } NM_i = \sum_{d=1}^{\lceil \frac{n}{2} \rceil} \left[3d \times \binom{\lfloor \frac{n}{2} \rfloor + d}{d}\right]$$

$$\text{Max } NM_i = \sum_{d=1}^{\lceil \frac{n}{2} \rceil} \left[\left(3d \times \left\lceil \frac{n}{2} \right\rceil + d + 3\right) \times \binom{\lfloor \frac{n}{2} \rfloor + d}{d}\right]$$

3.2 Bae의 연구

Bae의 알고리즘[1]은 하이퍼큐브에 삽입된 논리적 순환 메쉬를 기초로 공유자원에 접근하려는 요청노드 i의 요청집합 S_i를 노드 i와 동일한 행과 열에 있는 모든 노드들로 정의한다. 따라서 하이퍼큐브에서 공유자원 접근 요청노드 $a_{n-1}a_{n-2}\ldots a_2a_1a_0$의 요청집합은 *가 don't care bit일 때 $S_{a_{n-1}a_{n-2}\ldots a_2a_1a_0} = \{(*\cdots *|a_{\lceil n/2 \rceil -1} a_{\lceil n/2 \rceil -2}\ldots a_1a_0), (a_{n-1}a_{n-2}\ldots a_{\lceil n/2 \rceil}|*\cdots *)\}$가 된다. 공유자원 요청노드는 자신과 같은 행과 열에 있는 노드들에게 REQUEST 메세지를 전송한다. 메세지를 받은 요청집합 내의 노드들은 자원에 접근할 수 있도록 인가해주기 위해 GRANT 메세지를 보내거나 공유자원이 해제될 때까지 요청노드를 블럭시킨다. 요청노드 i가 S_i내 모든 노드로부터 GRANT 메세지를 받으면 공유자원에 접근하여 수행하고, 더 이상 공유자원이 필요하지 않게 되면 S_i내의 모든 노드들에게 RELEASE 메세지를 전송한다. 두 개의 요청노드 i와 j가 존재할 때 각 요청집합 S_i와 S_j에는 최소한 두 개의 공통된 노드가 존재하므로 이 공통된 노드가 조정자로서의 역할을 한다.

n-차원 하이퍼큐브 Q_n에서, 공유자원 접근 요청노드 i와 요청집합 S_i내의 노드들 간에 가장 먼 거리는 $\lceil n/2 \rceil$이고, S_i의 cardinality는 $|S_i| = 2^{\lceil n/2 \rceil} - 1$이다. 그리고 노드 i로부터 거리 d에 있는 S_i내의 노드 개수는 $2 \times \binom{\lceil n/2 \rceil}{d}$개 이다. 따라서 Bae의 알고리즘의 성능은 다음과 같다.

• 최소 왕복 지연:

$$D = 2 \times \left\lceil \frac{n}{2} \right\rceil$$

• 블럭킹 지연:

$$B = \frac{1}{N-1} \sum_{d=1}^{\lceil \frac{n}{2} \rceil} \left[2d \binom{\lfloor \frac{n}{2} \rfloor}{d} + 2 \left\lceil \frac{n}{2} \right\rceil \left(N - 2^{\lceil \frac{n}{2} \rceil + 1} \right) \right]$$

• 공유자원 접근당 메세지 개수:

$$\text{Min } NM_i = \sum_{d=1}^{\lceil \frac{n}{2} \rceil} \left[3d \times 2 \times \binom{\lfloor \frac{n}{2} \rfloor}{d} \right]$$

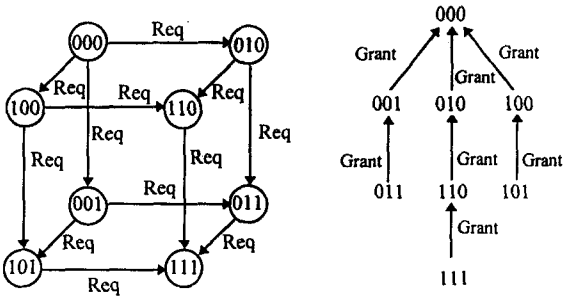
$$\text{Max } NM_i = \sum_{d=1}^{\lceil \frac{n}{2} \rceil} \left[7d \times 2 \times \binom{\lfloor \frac{n}{2} \rfloor}{d} \right]$$

3.3 Naimi의 연구

Raynal[14]은 분산 시스템에서 사용되는 상호배제 알고리즘을 인가-기반 알고리즘과 토큰-기반 알고리즘 두가지 그룹으로 분류하고 있다. Naimi[10]는 인가-기반 상호배제 알고리즘인 Ricart와 Agrawala의 알고리즘[12]과 토큰-기반 상호배제 알고리즘인 Naimi와 Trehel의 알고리즘[16]을 n-차원 하이퍼큐브 구조에 맞게 적용하여 성능을 비교하였다.

Naimi는 n-차원 하이퍼큐브에 Ricart의 알고리즘을 적용하여, 공유자원에 접근하고자 하는 n-차원 하이퍼큐브의 노드 x는 나머지 다른 모든 노드들에게 최적 flooding 방법[2]에 따라 REQUEST 메세지를 전송하고, 이 모든 노드로부터 공유자원 접근을 인가하는 GRANT 메세지를 받은 후에 공유자원에 접근한다. 노드 x로부터 REQUEST 메세지를 받은 노드는 임의의 최단 경로로 요청노드에게 즉각 GRANT 메세지를 전송 하거나, 공유자원이 해제될 때까지 요청노드를 블럭시킨다. REQUEST 메세지를 받은 노드 y 역시 공유자원에 접근하고자 하는 경우에는 요청노드 x와 y 자신 중 어느 쪽이 먼저 접근하도록 할 것인가를 즉시 계산하기 위해 REQUEST 메세지를 만든 노드에 관련된 순차번호(타임 스탬프)에 따라 우선순위를 비교한다.

$T_x[y]$는 노드 x가 노드 y로부터 받은 REQUEST 메세지의 마지막 순차번호를, $T_x[x]$는 요청노드 x의 마지막 순차번호를 나타낼 때, $(T_x[x] < T_x[y])$ 또는 $(T_x[x] = T_x[y],\ x < y)$라면 요청노드 x가 우선권을 가진다. (그림 4)는 Q_3의 노드 000이 공유자원 접근을 요청하여 전송하는 REQUEST 메세지(a)와 모든 다른 노드로부터 GRANT 메세지(b)를 받는 경로를 보여준다.



(그림 4) (a)요청노드 000의 REQUEST 메세지, (b)다른 노드들로부터 GRANT 메세지 수신 경로

(Fig. 4) (a)transmission of request meaasges, (b)the transmission of permission messages

n-차원 하이퍼큐브 Q_n에서 공유자원 접근 요청노드 i의 요청집합 S_i는 시스템내의 모든 다른 노드들을 포함하므로, S_i의 cardinality는 $|S_i| = 2^n - 1$이다. 그리고 요청노드 i와 S_i내의 노드간의 가장 먼 거리는 n이고, 요청노드 i로부터 거리 d에 있는 S_i내의 노드의 개수는 $\binom{n}{d} = \frac{n!}{d!(n-d)!}$ 이다. 따라서 Naimi의 알고리즘의 성능은 다음과 같다.

• 최소 왕복 지연:

$$D = n$$

• 블럭킹 지연:

$$B=\frac{1}{N-1}\left[\sum_{d=1}^{n} d\binom{n}{d}\right]$$

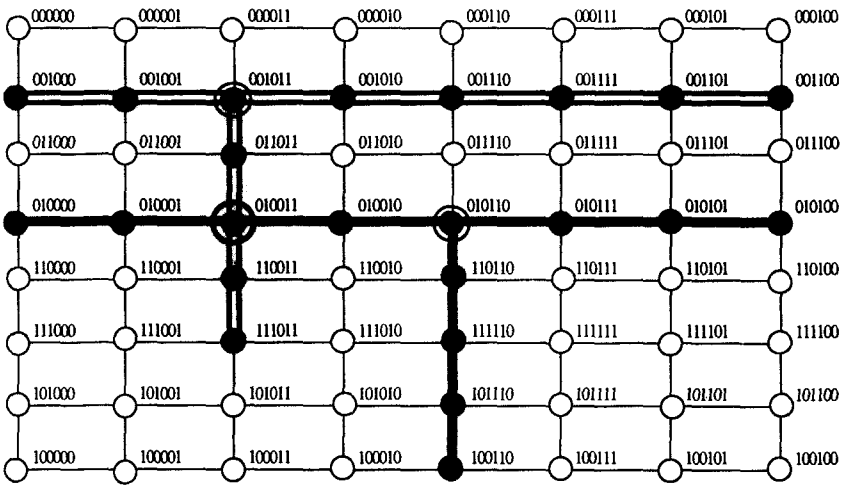
• 공유자원 접근당 메세지 개수:

$$NM_i=\sum_{d=1}^{n} d\binom{n}{d}=n\times 2^{n-1}$$

4. 분산 상호배제 알고리즘의 설계

본 논문에서는 하이퍼큐브 멀티컴퓨터 상에서 공유자원에 대한 상호배제를 효과적으로 해결하기 위한 인가-기반 분산 상호배제 알고리즘을 설계한다. 이 알고리즘은 상호배제를 해결하기 위해 2^{r+s}-차원 하이퍼큐브에 삽입되는 $2^r\times 2^s$ 논리적 순환 메쉬 구조로부터 공유자원 접근 요청노드 i가 REQUEST 메세지를 전송하고 GRANT 메세지를 수신해야 하는 효율적인 T-패턴의 요청집합 S_i를 구한다. 요청집합 Si는 논리적 순환 메쉬에서 노드 i와 같은 행에 속하는 모든 노드들(**T**-패턴의 − 부분)과 같은 열에 속하는 노드 중 노드 i로부터 하향으로 연속된 $2^{s-1}=2^{\lceil n/2\rceil-1}$개의 노드(**T**-패턴의 | 부분)로 구성된다. (r +s)-차원 하이퍼큐브에서 공유자원에 접근하고자 하는 노드 i는 **T**-패턴의 요청집합 S_i에 포함된 모든 노드에게 Lan의 멀티캐스트 방식[8]에 따라 REQUEST 메세지를 전송한다.

T-패턴 요청집합은 시스템의 임의의 두 노드 i, j에 대해 언제나 $S_i\cap S_j\neq\phi$로서 공통된 노드가 하나 이상 존재한다. 두 요청노드 i와 j가 $2^r\times 2^s$ 순환 메쉬에서 같은 행에 위치할 경우에는 적어도 그 행 전체 즉 2^s개의 공통노드가 존재한다. 두 요청노드가 같은 열에 위치할 때는 각 노드의 요청집합에는 자신과 같은 열의 하향 $2^{\lceil n/2\rceil-1}$개 노드가 포함되며 순환 메쉬에서의 두 노드간의 최대 거리는 $2^{\lceil n/2\rceil-1}$이다. 이 두 노드의 T-패턴 요청집합 내 동일 열에 속한 노드 개수를 합하면 각자 자신을 포함하여 $(1+2^{\lceil n/2\rceil-1})+(1+2^{\lceil n/2\rceil-1})=2+2^{\lceil n/2\rceil}$이다. 즉 두 요청집합간에는 자신이 포함된 적어도 2개의 공통된 노드가 존재한다. 두 요청노드가 서로 다른 행과 열에 위치할 때는, T-패턴 요청집합에는 메쉬의 한 행 전체가 포함되므로 i의 열과 j의 행이 교차하는 노드 또는 j의 열과 i의 행이 교차하는 노드로 인해 반드시 한 개 이상의 공통노드가 존재한다. REQUEST 메세지가 수신되는 임의의 노드는 REQUEST 메세지를 전송한 노드들 중 하나의 요청노드에게만 자원 접근을 인가하는 GRANT 메세지를



(그림 5) 요청노드 001011과 요청노드 010110의 요청집합
(Fig. 5) Request sets of node 001011 and node 010110

전송한다. 두 노드 i와 j의 요청집합 S_i와 S_j에는 최소한 한 개의 공통된 노드가 존재하고 공통노드는 단 하나의 요청노드에게만 GRANT 메세지를 전송하므로, 어느 한 순간에 두 노드가 동시에 같은 공유자원에 대한 인가를 받을 수는 없으므로 상호배제가 안전하게 보장된다.

(그림 5)는 Q_6의 노드 010110의 요청집합 S_{010110}={010000, 010001, ***010011***, 010010, 010110, 010111, 010101, 010100, 110110, 111110, 101110, 100110}와 노드 001011의 요청집합 S_{001011}={001000, 001001, 001011, 001010, 001110, 001111, 001101, 001100, 011011, ***010011***, 110011, 111011}를 보여준다. 이 두 요청집합 간에는 공통노드 010011이 존재한다. 이 공통노드 010011은 두 요청노드 010110와 001011간의 조정자 역할을 한다.

n-차원 하이퍼큐브 Q_n의 요청노드 $a_{n-1}a_{n-2}\ldots a_2a_1a_0$의 요청집합 $S_{a_{n-1}a_{n-2}\ldots a_2a_1a_0}$를 구하는 알고리즘은 (그림 6)과 같다. 여기서 십진수 i에 대응하는 n비트의 이진코드를 그레이 코드로 변환하는 함수 G(i, n)=$b_{n-1}b_{n-2}\ldots b_2b_1b_0$로 $b_{n-1}=a_{n-1}$, $b_{n-2}=a_{n-1}\oplus a_{n-2}$, ..., $b_2=a_3\oplus a_2$, $b_1=a_2\oplus a_1$, $b_0=a_1\oplus a_0$ 연산을 실행한다. 그레이 코드를 이진 코드로 변환하는 함수 GB($a_{n-1}a_{n-2}\ldots a_2a_1a_0$)=$b_{n-1}b_{n-2}\ldots b_2b_1b_0$로서 $b_{n-1}=a_{n-1}$, $b_{n-2}=b_{n-1}\oplus a_{n-2}$, ..., $b_1=b_2\oplus a_1$, $b_0=b_1\oplus a_0$ 연산을 실행한다. 함수 D($b_{n-1}b_{n-2}\ldots b_2b_1b_0$)는 이진코드 $b_{n-1}b_{n-2}\ldots b_2b_1b_0$를 십진값으로 변환하는 함수이다.

```
Procedure request_set(a_{n-1}a_{n-2}...a_2a_1a_0)
begin
  S_{a_{n-1}a_{n-2}...a_2a_1a_0} = {a_{n-1}a_{n-2}...a_2a_1a_0};
  for  0 ≤ i <   do
    begin
      GC_i = G(i, ⌈n/2⌉);
    end;
  for  0 ≤ i < ⌈n/2⌉  do
    begin
      S_{a_{n-1}a_{n-2}...a_2a_1a_0} = S_{a_{n-1}a_{n-2}...a_2a_1a_0} ∪ ( a_{n-1}a_{n-2}...a_{⌈n/2⌉} | GC_i );
    end;
  j = D(GB(a_{⌈n/2⌉-1}...a_1a_0));
  for  0 ≤ i < 2^{⌈n/2⌉-1}  do
    begin
      j = (j + 1);
      k = j mod 2^{⌈n/2⌉};
      S_{a_{n-1}a_{n-2}...a_2a_1a_0} = S_{a_{n-1}a_{n-2}...a_2a_1a_0} ∪ (GC_k | a_{⌈n/2⌉-1}...a_1a_0);
    end;
end;
```

(그림 6) 노드 $a_{n-1}a_{n-2}\ldots a_2a_1a_0$의 요청집합 $S_{a_{n-1}a_{n-2}\ldots a_2a_1a_0}$를 구하는 알고리즘

(Fig. 6) Request set algorithm for a node $a_{n-1}a_{n-2}\ldots a_2a_1a_0$

n-차원 하이퍼큐브 Q_n에서, 본 논문의 상호배제 알고리즘이 사용하는 정보구조인 T-패턴 요청집합의 cardinality, $|S_{a_{n-1}a_{n-2}\ldots a_2a_1a_0}|$는 다음과 같다.

$$|S_{a_{n-1}a_{n-2}\ldots a_2a_1a_0}| = 2^{\lceil \frac{n}{2} \rceil} + \frac{2^{\lceil \frac{n}{2} \rceil}}{2} + 1 - 1$$
$$= 2^{\lceil \frac{n}{2} \rceil} + 2^{\lceil \frac{n}{2} \rceil - 1}$$
$$= 3 \cdot 2^{\lceil \frac{n}{2} \rceil - 1}$$

본 논문에서 제안하는 알고리즘은 Sanders의 알고리즘[13]과 같은 프로토콜을 사용하며, Q_n에서 공유자원에 접근하려는 요청노드 i는 다음과 같은 프로토콜을 실행한다.

- REQUEST: 노드 i는 타임 스탬프를 갖는 REQUEST 메세지, REQ(i)를 요청집합 S_i내의 모든 노드들에게 Lan의 멀티캐스트 방식에 따라 전송하고 모든 노드 $j \in S_i$로부터 공유자원 접근을 인가하는 GRANT 메세지, GRANT(j)를 기다린다.
- ACCESS: 요청집합 S_i내의 모든 노드로부터 공유자원 접근을 인가하는 GRANT 메세지를 받으면 공유자원에 접근하여 수행한다.
- RELEASE: 공유자원이 더 이상 필요 없을 때는 공유자원을 해제하는 RELEASE 메세지, REL(i)를 요청집합 S_i내의 모든 노드들에게 Lan의 멀티캐스트 방식에 따라 전송한다.

요청노드 i의 요청집합에 포함되는 노드 j는 다음과 같은 프로토콜을 실행한다.

- GRANT: 공유자원을 요청하는 노드 i에게 접근을

인가할 수 있다면 요청노드 i로의 최단 경로를 따라 GRANT 메세지, GRANT(j)를 보낸다.

- FAIL : 방금 도착한 노드 i의 REQ(i)의 타임 스탬프가 이미 공유자원 접근을 인가하는 GRANT(j)를 전송한 다른 요청노드 k의 타임 스탬프보다 느린 경우에는, 요청노드 i까지의 최단 경로로 FAIL 메세지, FAIL(j)를 전송한다.
- INQUIRE : 이미 특정 노드 k에게 공유자원 접근을 인가하는 GRANT(i)를 보낸 후에 노드 i로부터 받은 REQ(i)의 타임 스탬프가 REQ(k)보다 더 빠른 경우에는 노드 k에게 인가한 GRANT(j)에 대한 취소를 의뢰하는 INQUIRE 메세지, INQ(j)를 노드 k로의 최단 경로로 전송한다. INQ(j)를 받은 노드 k는 노드 j로부터 받은 GRANT(j)를 폐기하고 YIELD 메세지, YIELD(k)를 노드 j로의 최단 경로로 전송한다.
- New GRANT : 노드 k로부터 YIELD(k)를 받은 노드 j는 REL(k)를 받은 것처럼 GRANT(j)를 요청노드 i에게 최단 경로로 보낸다. 이후에 노드 i로부터 REL(i)를 받으면 이전에 INQ(j)를 보내서 YIELD(k)를 보낸 노드 k에게 새로 GRANT(i)를 최단 경로로 보낸다.

n-차원 하이퍼큐브 Q_n의 각 노드 i는 다음과 같은 알고리즘을 수행하게 된다.

- 노드 i의 지역 변수 및 자료 구조

  CSSTAT : 노드 i가 알고 있는 공유자원의 상태로서, 공유자원이 접근 가능하다면 free를 그렇지 않다면 노드 i가 GRANT 메세지를 전송하여 아직 RELEASE 메세지를 받지 않은 노드 즉 공유자원을 사용 중인 노드의 ID를 가진다.

  Q_REQ(i) : 노드 i에 도착한 임의의 노드 $j \in Q_n$의 REQ(j)를 타임 스탬프 T_j순으로 저장하고 있는 큐

- REQ(j)를 수신한 경우

  1. Q_REQ(i) ← REQ(j);
  2. *if* CSSTAT ≠ free *then*
     if $T_{CSSTAT} < T_j$ *then*
     FAIL(i)를 j로 전송;
     else
     {*if* 이미 노드 CSSTAT에게 INQUIRE(i)를 보내지 않았다면 *then*
     INQ(i)를 노드CSSTAT에 전송;
     if $\forall k \in$ Q_REQ(i), $T_k > T_j$이고 노드 k에게 FAIL(i)를 전송하지 않았다면 *then*
     FAIL(i)를 k로 전송;}
     else
     {GRANT(i)를 Q_REQ(i)의 탑 노드 t에게 전송;
     노드 t를 Q_REQ(i)에서 제거;
     CSSTAT ← 노드 t;}

- REL(j)를 수신한 경우

  1. CSSTAT ← free;
  2. *if* Q_REQ(i) ≠ ϕ *then*
     {$Q_REQ(i)_1$의 맨 앞에 있는 노드 t에게 GRANT(i)를 전송;
     CSSTAT ← 노드 t;}

- INQ(j)를 수신한 경우

  1. $\forall k \in S_i$로부터 수신한 메세지를 점검;
  2. *if* FAIL(k)를 수신했거나 YIELD(i)를 임의의 노드k에게 전송한 후 새로 GRANT(k)를 받지 않았다면 *then*
     {노드 j로부터 수신한 GRANT(j)를 폐기;
     YIELD(i)를 노드 j에게 전송;}

- YIELD(j)를 수신한 경우

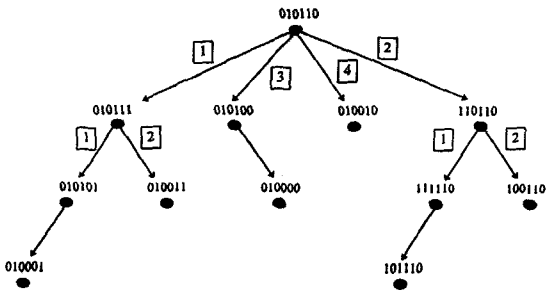
  1. CSSTAT ← free;
     노드 j의 REQ(j)를 다시Q_REQ(i)에 넣기;
  2. REL(j)를 수신한 것과 같이 진행함;

(그림 5)의 노드 010110가 공유자원을 요청하면 T-패턴에 의해 요청집합 S_{010110}안의 모든 노드들에게 Lan의 멀티캐스트 방식에 의해 REQUEST 메세지를 보낸다. Lan의 멀티캐스트 방식에서 출발점 노드는 요청노드 010110가 되고, 목적지 노드 리스트는 요청집합 S_{010110}={010000, 010001, 010011, 010010, 010110, 010111, 010101, 010100, 110110, 111110, 101110, 100110}가 된다. 출발점 노드에서 보내는 REQUEST 메세지에는 출발점 노드의 레이블, 목적지 노드 리스트, 단위 데이터를 포함한다. (그림 7)은 Lan의 방식에 의해 메세지가 전송되는 과정 중 출발점 노드에서 목적지 노드 리스트 안의 이웃 노드로 REQUEST 메세지를 멀티캐스트하는 과정을 보인다.

참조배열은 목적지 노드 리스트 내의 각 노드에 대해 노드 레이블 ⊕출발점 노드 레이블에 의해 얻어진 집합으로 {010110, 010000, 010001, 010011, 010010, 010111, 010101, 010100, 110110, 111110, 101110, 100110}이 된다. 이 배열의 각 원소값에 포함된 '1'의 개수가 곧 출발점 노드와의 해밍 거리를 나타낸다. 출발점 노드는 멀티캐스트하기 위해 참조배열의 각 원소에 대해 비트값 '1'을 가장 많이 갖는 비트 위치 $l(=0)$을 찾는다. 이 비트위치 $l=0$에 해당하는 비트값이 1인 목적지 서브리스트-1={010001, 010011, 010111, 010101}은 메세지 헤드에 첨가되어, 비트위치 $l=0$에 해당하는 차원 d_0를 따라 이웃 노드 010111로 전송한다. 이 때 참조배열 원소 중 이 서브리스트에 포함된 참조값은 0으로 세트된다. 2단계에서는 1단계에서 변경된 참조배열에서, '1'값이 가장 많은 비트위치 $l=5$이고 목적지 서브리스트-2={110110, 111110, 101110, 100110}를 메세지 헤드에 첨가한 REQUEST 메세지를 차원 d_5에 연결된 이웃노드 110110에게 전송한다. 역시 목적지 서브리스트-2에 해당하는 참조값은 참조배열에서 0으로 세트된다. 3단계로 '1'의 값이 가장 많은 비트위치 $l=1$이고 목적지 서브리스트-3={010000, 010100}를 첨가한 REQUEST 메세지는 차원 d_1에 연결된 이웃노드 010100에게 전송된다. 4단계로 현재 참조배열은 000100만을 제외하고 모두 0으로 세트되었으므로 목적지 서브리스트-4={010010}를 메세지 헤드에 첨가한 REQUEST 메세지가 차원 d_2에 연결된 이웃노드 010100에게 전송된다.

REQUEST 메세지가 요청집합내의 노드들에게 멀티캐스트되는 중간 과정에서 REQUEST 메세지를 수신한 노드는 자신의 주소와 수신 메세지에 포함된 목적지 서브리스트의 주소를 비교한다. 만일 일치하는 주소가 있다면, 자신에게 온 메세지이므로 목적지 서브리스트에서 그 주소를 제거하고 데이터 필드의 사본을 로컬 처리기에게 보낸다. 일치한 주소를 제거한 후 목적지 서브리스트가 공집합이 되면, 더 이상의 메세지 전송은 일어나지 않는다. 목적지 서브리스트가 공집합이 아니라면 수신 노드 자신이 출발점 노드로서, 현재의 목적지 서브리스트를 가지고서 같은 방식으로 멀티캐스트를 수행한다.

(그림 7)의 예에서 알 수 있듯이, Lan의 멀티캐스트 방식으로 요청노드 i에서 요청집합 S_i로 메세지를 전송하는 경우, 전체 발생하는 메세지 개수는 $|S_i|-1$이므로 $3\cdot 2^{\lceil n/2\rceil}-1$이다.



(그림 7) 출발점 노드 010110에서 목적지 노드 리스트 S_{010110}로의 Lan의 멀티캐스트 예

(Fig. 7) An example of Lan's multicast from source node 010110 to destination node list S_{010110}

5. 성능 평가

본 논문에서 제시한 상호배제 알고리즘의 성능을 3가지 성능 척도[5] 즉, 최소 왕복 지연, 블럭킹 지연, 공유자원 접근당 메세지 개수로써 평가하고, Gupta의 알고리즘, Naimi의 알고리즘, Bae의 알고리즘과 비교한다.

5.1 최소 왕복 지연

최소 왕복 지연은 요청노드 i가 공유자원을 요청한 순간부터 접근 인가를 받기까지 기다려야하는 최소 시간이다. 요청노드 i의 최소 왕복 지연 D_i는 다른 노드와 공유자원에 대한 경쟁이 없을 때 얻어진다. 요청노드 i와 S_i내의 노드들과의 최대 거리는 $d_{max}=\lceil n/2\rceil$이다. 노드 i는 최대 d_{max} 거리에 있는 노드와 REQUEST 메세지, GRANT 메세지를 주고 받으므로 $D_i=2\times\lceil n/2\rceil$이다. 모든 임의의 노드 i에 대해서도 D_i는 동일하다. 최소 왕복 지연 D는 모든 D_i의 최대값이므로 $D=2\times\lceil n/2\rceil$가 된다. 〈표 1〉은 각 알고리즘별 최소 왕

〈표 1〉 최소 왕복 지연의 비교

〈Table 1〉 Comparison of minimum round-trip delay for each algorithm

알고리즘 / 최소왕복지연	Gupta	Naimi	Bae	Our
D	$\lceil n/2\rceil$	n	$\lceil n/2\rceil$	$\lceil n/2\rceil$

복 지연을 보여 준다. Naimi의 알고리즘의 최소 왕복 지연은 요청집합이 다른 모든 노드들을 포함하므로 다른 알고리즘의 최소 왕복 지연의 두배가 된다.

5.2 블럭킹 지연

블럭킹 지연 B_{ij}는 노드 j가 자원을 해제한 후 블럭된 노드 i가 공유자원에 접근하기 전까지 요구되는 전송 메세지의 최대 개수이다. 노드 i의 요청집합에 속하는 노드 j는 두 노드간의 거리가 d일 때, j가 자원을 해제하면 j에서 i까지 GRANT 메세지가 도착하는데 d 단위 시간이 걸리므로 노드 $j \in S_i$, $i \neq j$에 대한 $B_{ij} = d$이다. 본 논문에서 제안하는 알고리즘의 요청집합 S_i에는 하이퍼큐브에 삽입된 논리적 메쉬에서 요청노드 i와 동일한 행의 모든 노드와 동일한 열에 있는 노드 중의 반을 포함한다. 노드 i와 같은 행에 있는 모든 노드 j와의 블럭킹 지연은

$$\sum_{d=1}^{\lceil \frac{n}{2} \rceil} d \binom{\lceil \frac{n}{2} \rceil}{d}$$

이며, 같은 열에 있는 노드들 중 반에 해당하는 j와의 블럭킹 지연은

$$\left(\frac{2^{\lceil \frac{n}{2} \rceil - 1}}{2^{\lceil \frac{n}{2} \rceil} - 1} \right) \sum_{d=1}^{\lceil \frac{n}{2} \rceil} \left[d \binom{\lceil \frac{n}{2} \rceil}{d} \right]$$

이다. 그러므로 모든 $j \in S_i$에 대한 B_{ij}의 합

$$B_i \approx \left(1 + \frac{2^{\lceil \frac{n}{2} \rceil - 1}}{2^{\lceil \frac{n}{2} \rceil} - 1} \right) \sum_{d=1}^{\lceil \frac{n}{2} \rceil} \left[d \binom{\lceil \frac{n}{2} \rceil}{d} \right]$$

이다.

이에 반해 노드 j가 노드 i의 요청집합에 속하지 않는 경우에는, j의 RELEASE 메세지가 노드 $k \in S_i$에게 도착하면, 노드 k는 i에게 GRANT 메세지를 보내게 된다. j와 k의 최대거리, k와 i의 최대거리 둘 다 $\lceil n/2 \rceil$이므로 최악의 경우 $B_{ij} = 2 \times \lceil n/2 \rceil$이다. 요청집합 S_i에 속하지 않는 노드의 개수는 $(N - 3 \cdot 2^{\lceil n/2 \rceil - 1})$이므로 노드 $j \notin S_i$에 대한 노드 i의 블럭킹 지연은 최악 경우,

$$B_i \approx 2 \lceil \frac{n}{2} \rceil \left(N - 3 \cdot 2^{\lceil \frac{n}{2} \rceil - 1} \right)$$

이고, 평균 경우

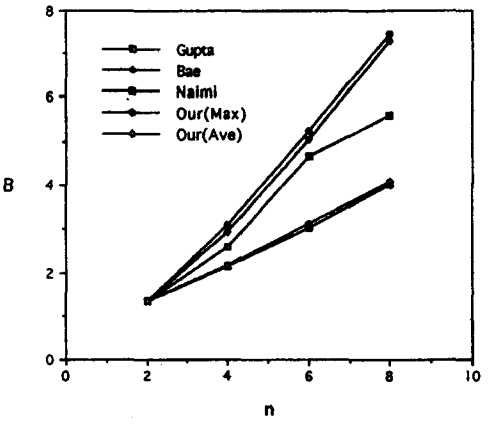
$$B_i \approx 2 \frac{\left(1 + \frac{2^{\lceil n/2 \rceil - 1}}{2^{\lceil n/2 \rceil} - 1} \right) \sum_{d=1}^{\lceil n/2 \rceil} \left[d \binom{\lceil n/2 \rceil}{d} \right]}{3 \cdot 2^{\lceil n/2 \rceil - 1}} \left(N - 3 \cdot 2^{\lceil \frac{n}{2} \rceil - 1} \right)$$

이다. Q_n에서 B_i는 모든 i에 대해 동일하므로 본 논문에서 제안하는 알고리즘의 최대 블럭킹 지연은

$$\text{Max } B \approx \frac{1}{N-1} \left[\left\{ \left(1 + \frac{2^{\lceil \frac{n}{2} \rceil - 1}}{2^{\lceil \frac{n}{2} \rceil} - 1} \right) \sum_{d=1}^{\lceil n/2 \rceil} \left(d \binom{\lceil \frac{n}{2} \rceil}{d} \right) \right\} + 2 \lceil \frac{n}{2} \rceil \left(N - 3 \cdot 2^{\lceil \frac{n}{2} \rceil - 1} \right) \right]$$

이고, 평균 블럭킹 지연은

$$\text{Ave } B \approx \frac{1}{N-1} \left[\left(1 + \frac{2^{\lceil n/2 \rceil - 1}}{2^{\lceil n/2 \rceil} - 1} \right) \sum_{d=1}^{\lceil n/2 \rceil} \left[d \binom{\lceil n/2 \rceil}{d} \right] + 2 \frac{\left(1 + \frac{2^{\lceil n/2 \rceil - 1}}{2^{\lceil n/2 \rceil} - 1} \right) \sum_{d=1}^{\lceil n/2 \rceil} \left[d \binom{\lceil n/2 \rceil}{d} \right]}{3 \cdot 2^{\lceil n/2 \rceil - 1}} \left(N - 3 \cdot 2^{\lceil \frac{n}{2} \rceil - 1} \right) \right]$$



(그림 8) 알고리즘별 블럭킹 지연 비교
(Fig. 8) Blocking delay for each algorithm

이다.

(그림8)은 각 알고리즘별 블럭킹 지연을 보여준다. 본 논문에서 제안하는 알고리즘의 최대 블럭킹 지연은 Bae의 알고리즘과 거의 같고, 평균 블럭킹 지연은 가장 우수한 Naimi의 알고리즘과 거의 같음을 알 수 있다.

5.3 공유자원 접근당 메세지 개수

공유자원 접근당 메세지 개수 NM_i는 요청노드 i가 공유자원에 접근하여 해제하기 까지 생성되는 메세지 개수이다. NM_i는 공유자원에 대한 경쟁이 없을 때 최소이고, 경쟁이 있을 때 최대이다. 최소 NM_i는 노드 i와 $j \neq i$인 노드 $j \in S_i$간에 3개의 메세지 (REQUEST, GRANT, RELEASE)가 교환될 때 발생한다. 따라서 본 논문에서 제안하는 알고리즘의 공유자원 접근당 최소 메세지 개수 Min NM_i는 다음과 같다.

$$\text{Min } NM_i \approx 2\cdot(3\cdot 2^{\lceil \frac{n}{2} \rceil - 1} - 1) + \sum_{d=1}^{\lceil n/2 \rceil} \binom{\lceil \frac{n}{2} \rceil}{d} + \frac{2^{\lceil \frac{n}{2} \rceil - 1}}{2^{\lceil \frac{n}{2} \rceil} - 1} \sum_{d=1}^{\lceil \frac{n}{2} \rceil} \left[d \binom{\lceil \frac{n}{2} \rceil}{d} \right]$$

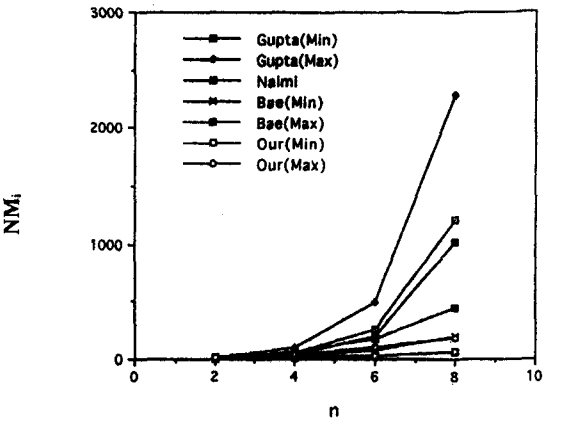
$$= 2\cdot(3\cdot 2^{\lceil \frac{n}{2} \rceil - 1} - 1) + \left(1 + \frac{2^{\lceil \frac{n}{2} \rceil} - 1}{2^{\lceil \frac{n}{2} \rceil} - 1}\right) \sum_{d=1}^{\lceil n/2 \rceil} \left[d \binom{\lceil \frac{n}{2} \rceil}{d} \right]$$

최대 NM_i는 노드 $j \notin S_i$에서 경쟁이 일어난 경우, $k \neq i$이며 $k \in (S_i, S_j)$인 공통노드 k와 i, j간에 (REQUEST, INQUIRE, FAIL, YIELD, GRANT, RELEASE, new GRANT)를 교환할 때 발생한다. 따라서 본 논문에서 제안하는 알고리즘의 공유자원 접근당 최대 메세지 개수 Max NMi는 다음과 같다.

$$\text{Max } NM_i \approx 2\cdot(3\cdot 2^{\lceil \frac{n}{2} \rceil - 1} - 2) + 4 \sum_{d=1}^{\lceil \frac{n}{2} \rceil} \left[d \binom{\lceil \frac{n}{2} \rceil}{d} \right]$$

$$+ 4 \left(\frac{2^{\lceil \frac{n}{2} \rceil - 1}}{2^{\lceil \frac{n}{2} \rceil} - 1}\right) \sum_{d=1}^{\lceil \frac{n}{2} \rceil} \left[d \binom{\lceil \frac{n}{2} \rceil}{d} \right]$$

$$= 2\cdot(3\cdot 2^{\lceil \frac{n}{2} \rceil - 1} - 1) + 4\left(1 + \frac{2^{\lceil \frac{n}{2} \rceil - 1}}{2^{\lceil \frac{n}{2} \rceil} - 1}\right) + \sum_{d=1}^{\lceil \frac{n}{2} \rceil} \left[d \binom{\lceil \frac{n}{2} \rceil}{d} \right]$$



(그림 9) 공유자원 접근당 메세지 개수
(Fig. 9) The number of messages for each algorithm

(그림 9)는 각 알고리즘에 대한 Min NM_i와 Max NM_i를 보여준다. 본 논문에서 제안한 알고리즘의 공유자원 접근당 요구되는 메세지 개수가 가장 적음을 알 수 있다.

6. 결 론

우리는 하이퍼큐브에 삽입된 메쉬를 기초로 T-패턴의 요청집합을 동적 자료 구조로 사용하는 분산 상호배제 알고리즘을 설계하고, 그것의 성능을 분석하고, 그리고 Gupta, Naimi, Bae의 알고리즘들과 성능을 비교하였다. 그 결과 본 논문에서 제안하는 알고리즘의 성능이 다른 알고리즘에 비해 최소 왕복 지연과 공유자원 접근당 메세지 개수 등이 우수함을 알 수 있었다. 그리고 본 논문에서 제안하는 알고리즘은 하이퍼큐브의 차원을 알면, 요청노드 i의 요청집합 S_i를 국부적으로 쉽게 계산할 수 있으며 n-차원 하이퍼큐브에서 요청노드 i와 요청집합 S_i내의 노드들과의 최대 거리는 $\lceil n/2 \rceil$이므로, 공유자원 접근을 위해 요청노드 i가 전송하는 메세지의 최대 전송 거리는 $\lceil n/2 \rceil$이다. 따라서 본 논문에서 제안하는 알고리즘은 특히 분산 실시간 시스템에 유용하다.

향후 연구 내용은 요청집합 S_i내의 노드에서 고장이 발생하더라도 상호배제를 보장할 수 있는 하이퍼큐브를 위한 고장 감래 분산 상호배제 알고리즘의 설

계와 하이퍼큐브를 위한 토큰-기반 상호배제 알고리즘의 설계 등이다.

참 고 문 헌

[1] I. H. Bae, "An Efficient Mutual Exclusion Algorithm for Hypercube Multicomputers", The transactions of the Korea information processing society, Vol. 3, No. 5, pp. 1207-1214, Sept. 1996.

[2] Bertsekas D. P., Özveren C., Stamoulis G. D., Tseng P., and Tsitsiklis J. N., "Optimal Communication Algorithms for Hypercubes", Journal of Parallel and Distributed Computing, Vol. 11, No. 4, pp. 263-275, 1994.

[3] S. J. Ha, Y. J. Kim, and I. H. Bae, "A Mutual Exclusion Algorithm Adapted to a Hypercube Architecture", Proceedings of ICOIN-9, pp. 449-454, Dec. 1994.

[4] S. J. Ha, H. T. Roh, S. K. Chun, and I. H. Bae, "A Fault Tolerant Distributed Mutual Exclusion Algorithm for Hypercube Multicomputers", Proceedings of HNTTI-1, pp. 127-133, July 1996.

[5] A. Gupta, S. C. Bruell, and S. Ghosh, "Mutual Exclusion on a Hypercube", Journal of Parallel and Distributed Computing 17, pp. 327-336, 1993.

[6] V. Kumar, A. Grama, A. Gupta,and G. Karypis, 'Introduction to Parallel Computing', The Benjamin/Cummings Pub. Co., Inc., 1994.

[7] L. Lamport, "Time, Clocks, and the Ordering of Events in Distributed System", Comm. of the ACM, Vol. 21, No. 7, pp. 558-565, July 1978.

[8] Y. Lan, H. II Esfahanian, and L. Ni, "Multicast in Hypercube Multiprocessors", Journal of Parallel and Distributed Computing, Vol. 8, pp. 30-41, 1990.

[9] M. Maekawa, "A Sqrt(N) Algorithm for Mutual Exclusion in Decentralized Systems", ACM Trans. on Computer Systems, Vol. 3, No. 2, pp. 145-159, May 1985.

[10] M. Naimi, "Distributed Mutual Exclusion on Hypercubes", Operating Systems Review, Vol. 30, No. 3, pp. 46-51, July 1996.

[11] P. Ramanathan and K. G. Shin, "Reliable Broadcast in Hypercube Multicomputers", IEEE Trans Computer Systems, Vol. 37, No. 12, pp. 1654-1657, Dec. 1988.

[12] G. Ricart and A. Agrawala, "An Optimal Algorithm for Mutual Exclusion in Computer Networks", Comm. ACM, Vol. 24, No. 1, pp. 9-27, Jan. 1981.

[13] B. A. Sanders, "The Information Structure of Distributed Mutual Exclusion Algorithms", ACM Trans. on Computer Systems, Vol. 5, No. 1, pp. 284-299, August 1987.

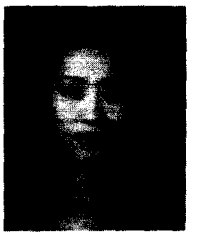
[14] M. Raynal, "A Simple Taxonomy for Distributed Mutual Exclusion Algorithms", Operating Systems Review, ACM Press, pp. 47-49, 1991.

[15] I. Suzuki and T. Kasami, "A Distributed Mutual Exclusion Algorithm", ACM Trans. Computer Systems, Vol. 3, pp. 344-349, 1985.

[16] M. Trehel and M. Naimi, "A Distributed Algorithm for Mutual Exclusion based on Data Structures and Fault Tolerance", Proc. IEEE Phoenix Conf. on Cop. and Comm., pp. 256-276, 1987.

[17] G. Cao, M. Singhal, Y. Deng, N. Rishe and W. Sun, "An Efficient Coterie-Based Mutual Exclusion Scheme With Fault-tolerance Capability", The Ohio State University, Technical Report TR 43, 1996.

[18] M. Singhal, "A Taxonomy of Didtributed Mutual Exclusion", Journal of Parallel and Distributed Computing, Vol. 18, pp. 94-101, 1993.



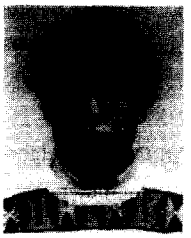
하 숙 정

1988년 제명대학교 전자계산학과 졸업(공학사)

1990년 중앙대학교 대학원 전자계산학과 졸업(공학석사)

1994년~1997년 대구효성가톨릭대학교 대학원 전산통계학과 전산학전공 박사과정 수료

관심분야: 분산 시스템(분산상호배제, 프로세서할당), 멀티미디어 시스템(주문형 비디오 시스템, 미디어 스케일링)



배 인 한

1984년 경남대학교 전자계산학과 졸업(공학사)

1986년 중앙대학교 대학원 전자계산학과 졸업(이학석사)

1990년 중앙대학교 대학원 전자계산학과 졸업(공학박사)

1996년~1997년 Department of Computer and Information Science, The Ohio State University, USA 박사후 과정

1989년~현재 대구효성가톨릭대학교 전자정보공학부 부교수

관심분야: 운영체제, 분산 시스템, 멀티미디어 시스템(특히, 고성능 주문형 비디오 시스템), 모빌 컴퓨팅