韓國컴퓨터情報學會 論文誌
第15卷 第3號, 2010. 3.

2010-15-3-3-3

연결 지향 네트워크에서 최초 가용 구간을 찾는 효율적인 알고리즘

정 균 락*

An Efficient Algorithm for Finding the Earliest Available Interval on Connection-Oriented Networks

Kyun-Rak Chong*

요 약

통신 기술과 저장 매체 발달로 인하여 광역 네트워크상에서 대규모 용량의 파일 전송이 요구되는 e-Science와 상업적 분야에서 다양한 응용이 가능하게 되었다. 이러한 대규모 데이터 전송을 위해서는 스케줄 가능한 높은 대역폭과 낮은 대기 시간이 요구되는 데 현재의 공용 인터넷에서는 예측 가능하고 효율적인 서비스를 제공하기 어렵다. 특히 데이터 제공자와 사용자가 지역적으로 떨어져 있을 때에는 이러한 원격 작업을 효율적으로 지원하기 위하여 제공자와 사용자간에 전용 연결이 필요하게 된다. 현재 전용 연결을 위한 여러 연구 프로젝트들이 진행되고 있다. 이러한 네트워크들의 관리시스템은 대역폭 스케줄러가 탑재되어 있는 데, 스케줄러는 네트워크 형태 정보와 링크의 대역폭 할당 정보를 토대로 하여 사용자 요구에 의해 경로를 계산한다. 본 논문에서는 전용 연결을 지원하는 네트워크에서 최소 대역폭과 필요한 전송 시간(duration)이 주어졌을 때 전송 가능한 가장 빠른 시간을 찾는 효율적인 알고리즘을 개발하고 기존의 알고리즘과 비교 분석하였다.

Abstract

The advancement of communication and networking technologies has enabled e-science and commercial application that often require the transport of large volume of data over wide-area network. Schedulable high-bandwidth low-latency connectivity is required to transport the large volume of data. But the public Internet does not provide predictable service performance. Especially, if data providers and users are far away, dedicated bandwidth channels are needed to support remote process efficiently. Currently several network research projects are in progress to develop dedicated connectionsy sA bandwidth scheduler computes an user requested path based on network topology information and link bandwidth allocationsy In this paper, we have proposed an efficient algorithm for finding the earliest time interval when minimum bandwidth and duration are giveny Our algorithm is experimentally compared with the known algorithm.

▸ Keyword : 전용연결(dedicated connection), 대역폭 스케줄링(bandwidth scheduling), 연결 지향 네트워크(connection-oriented network)

• 제1저자 : 정균락
• 투고일 : 2010. 01. 08, 심사일 : 2010. 01. 25, 게재확정일 : 2010. 03. 12.
* 홍익대학교 컴퓨터공학과 교수
※ 이 논문은 2008학년도 홍익대학교 학술연구진흥비에 의하여 지원되었음

Ⅰ. 서 론

통신 기술과 저장 매체 발달로 인하여 광역 네트워크상에서 대규모 용량의 파일 전송이 요구되는 e-Science와 상업적 분야에서의 응용이 가능하게 되었다. 이러한 응용 분야로는 고 에너지 핵 물리, 전파 천문학, 지구 과학, 시각화를 위한 이미지, 은행의 거래 레코드 보관 등이 있다. 예를 들어 고 에너지 핵물리 분야에 사용되는 데이터의 크기는 현재는 petabytes(1015)이지만 2015년경에는 exabytes(1018)가 될 것으로 예상된다 [1]. 이러한 대규모 데이터 전송을 위해서는 스케줄 가능한 높은 대역폭과 낮은 대기 시간이 요구되는 데 현재의 공용 인터넷에서는 예측 가능하고 효율적인 서비스를 제공하기 어렵다. 특히 데이터 제공자와 사용자가 지역적으로 떨어져 있을 때에는 이러한 원격 작업을 효율적으로 지원하기 위하여 제공자와 사용자간에 전용 연결(dedicated connection)이 필요하게 된다 [2, 3]. 이러한 전용 연결 성능을 가진 네트워크 연구 프로젝트들이 이미 진행되고 있으며 [4, 5, 6, 7, 8, 9, 10, 11], 국가 또는 국제적인 차원의 네트워크인 Internet2와 LHCNet도 배치되고 있다 [12, 13]. 이러한 네트워크들의 관리 시스템은 대역폭 스케줄러(bandwidth scheduler)가 탑재되어 있는 데, 대역폭 스케줄러는 네트워크 형태 정보와 링크의 대역폭 할당 정보를 토대로 하여 사용자 요구에 의해 경로를 계산한다. 그 다음 그 경로를 따라 링크들의 대역폭 할당 정보가 갱신되고 경로가 형성된다.

출발지와 목적지가 주어졌을 때 이 두 지점의 전용 연결을 위한 여러 문제들에 대한 연구가 진행되고 있는 데, 최초 구간 계산 문제는 최소 대역폭과 필요한 전송 시간(duration)이 주어졌을 때 출발노드에서 목적노드로 가는 전송 가능한 가장 빠른 시간 구간(slot)을 찾는 문제이다. 이 문제를 해결하는 방법으로 모든 가능한 시간 구간을 정렬해 놓고, 각 시간 구간에 대해 하나씩 너비 우선 탐색을 사용하여 경로를 찾는 알고리즘(이하 FS 알고리즘)이 제안되었는 데 [2], 최악의 경우에는 시간 구간의 총수만큼 너비 우선 탐색을 해야 하는 단점이 있다.

본 연구에서는 전용 연결을 지원하는 네트워크에서 최초 구간 계산 문제를 해결하는 효율적인 알고리즘(이하 BF 알고리즘)을 개발하고, FS 알고리즘[2]과 실험을 통해 비교 분석하였다. BF 알고리즘은 출발 노드로부터 목적 노드로 가는 경로를 찾는 데 있어 가장 빠른 시간 구간을 갖는 노드부터 방문하는 데, 일단 목적지에 도달하여 현재까지 가장 빠른 시간을 구한 뒤에는 이 시간 보다 늦은 시간 구간을 갖는 노드는 방문하지 않기 때문에 일반적으로 실행 시간이 빠르게 된다. 또 본 연구에서는 네트워크상의 노드의 수, 각 노드에서 링크의 수, 각 링크가 가지는 사용 가능한 시간 구간의 수, 출발노드와 목적노드사이의 평균 경로 길이를 고려한 다양한 데이터를 생성하여 BF 알고리즘과 FS 알고리즘의 성능을 비교하였다.

본 논문의 구성은 다음과 같다. 2 장에서는 관련 연구들을 알아 보고, 3 장에서는 제안하는 알고리즘에 관해 설명한다. 4 장에서는 실험 결과에 대해 분석하고 5장에서 결론을 맺는다.

Ⅱ. 관련 연구

전용 연결에는 두 가지 모드가 있는 데, 맞춤형(on-demand) 모드에서는 필요할 때 연결 요구가 이루어지고 현재 대역폭의 가용성에 따라 요구가 수용되거나 기각된다. 선점형(in-advance) 모드에서는 연결 요구가 대역폭 할당 스케줄에 따라 앞으로 사용 가능한 시간 구간에 배정된다.

DRAGON[4], CHEETA[6], JGN[10], HOPI[12], UCLP[14]은 맞춤형 모드를 지원하고, OSCARS[5], Enlighten[8], GeantII[9], USN[15]은 선점형 모드를 지원한다. 이 중 OSCARS는 예약 기능이 있으나 추적 루트(trace route)를 사용하기 때문에 모든 가능한 경로를 조사하지 못하며, 수학적으로 검증된 선점형 스케줄링 기능을 갖춘 네트워크는 USN이 처음이라고 할 수 있다[2].

전용 채널에서 출발 노드와 목적 노드가 주어졌을 때 다음 문제들에 대한 선점형 스케줄링 알고리즘들이 제안되었다 : (ⅰ) 대역폭이 최소한 b 이상이고 주어진 시작 시간과 종료시간을 만족하는 경로를 찾는 문제(Fixed Slot) (ⅱ) 네트워크 상에서 가장 큰 대역폭을 찾는 문제 (ⅲ) 대역폭과 필요한 전송 시간이 주어졌을 때 전송할 수 있는 가장 빠른 시간을 찾는 문제 (First Slot) (ⅳ) 모든 가능한 구간을 찾는 문제. 또, Fixed-Slot 알고리즘을 변형한 알고리즘들이 제시되었고 실험을 통해 비교 분석되었다 [3].

대용량 파일 전송을 위한 대역폭 스케줄링 관련 연구를 보면, e-science 응용을 위한 연결 지향 광학 네트워크에서 대용량 파일들의 전송을 위한 선점형 다중 경로 알고리즘들이 연구되었고, 여러 개의 출발 노드와 여러 개의 목적 노드 사이에 다수 개의 파일들을 전송한다고 할 때, 모든 파일의 전송 완료 시간을 최소로 하는 문제(Earliest Finish Time File Transfer Problem)를 위한 온라인 스케줄링 알고리즘과 주기적 배치(periodc batch) 알고리즘이 제안되었다 [16]. 광학 흐름 스위칭(optical flow switching)을 지원하는 광학 네트워크에서 동시 파일 전송 문제를 위한 제어 평면 알고리즘(control plane algorithm)을 개발되었는데 선형 계획법을 토대로 하고 있다.

동시 파일 전송 문제는 전송 시작 시간과 전송 완료 시간들이 주어진 다수의 벌크 파일 전송을 하는 데 있어 동시 전송 처리량을 높이는 것을 목적으로 하고 있다 [17]. 또, e-science 응용을 위한 네트워크에서 벌크 데이터 전송을 위한 승인 통제(admission control)와 대역폭 스케줄링 알고리즘을 제안하였다. 승인 통제는 전송될 벌크 데이터들은 일단 승인되면 각각의 전송 완료 시간 내에 전송 처리가 보장되도록 벌크 데이터들을 선택하는 것을 의미한다. 이 연구에서는 네트워크의 동시 전성 처리율을 높이기 위해 승인 통제, 전송 시간 예약, 다중 경로 라우팅, 대역폭 재할당에 기초한 최적화 방법을 제시하였다 [18]. 소요시간의 한계를 결정함으로써 요구 대역폭에 따른 서비스를 제공할 수 있는 동적 운용 프로토콜 및 효율적인 알고리즘도 제안되었다 [19c].

III. BF 알고리즘

1. 최초 구간 계산 문제

최초 구간 계산 문제에서 전용 연결 네트워크는 유향 그래프(directed graph) G = (V, E)로 표현할 수 있는 데 각 노드는 라우터와 같은 스위칭 장비를 나타내고 각 에지는 이 스위칭 장비들을 연결하는 링크이다. 각 에지에는 [2]에서와 같이 그 링크를 사용할 수 있는 시간 구간 리스트 $[s_1, e_1]$, $[s_2, e_2]$, …… , $[s_m, e_m]$을 가지고 있다고 가정한다. si는 구간의 시작 시간이고, ei는 구간의 종료시간이며, $s_1 < s_2 < \cdots\cdots < s_m$ 이다. 여기서 $[s_i, e_i]$는 이 구간안의 임의의 시점을 x라 할 때 x에서 x+T 시간동안 가용 대역폭이 최소한 b 이상이라고 가정하고 T는 필요한 전송시간이다.

그림 1에 최초 구간 계산 문제의 예가 나타나 있다. 이 예에서 1번 노드가 출발 노드이고 8번 노드가 목적노드이다. 가장 빠른 전용 연결 경로는 1-4-7-8인데 에지 <1, 4>, <4, 7>, <7, 8>의 시간 구간 리스트들의 교집합을 구하면 {[0, 2], [4, 6]} ∩ {[1, 3], [5, 7]} ∩ {[1, 5], [6, 7]} = {[1, 2]}가 되므로 사용 가능한 가장 빠른 시간은 1이다.

FS 알고리즘은 모든 시간 구간의 시작시간을 정렬해 놓고 각 시작시간에 대해 하나씩 모든 링크가 시작시간을 포함하는 시간 구간을 가지고 있는 경로를 찾기 위해 너비 우선 탐색을 실시한다. 그러므로 그림 1의 예에서 시작 시간을 정렬하면 $0 < 1 < \cdots\cdots < 8$이 되고, 시작시간 0부터 0을 포함하는 시간 구간을 가지고 있는 경로가 있는 지 너비 우선 탐색으로 찾아본다. 가능한 경로가 없으므로 다음 시작시간인 1에 대해 같은 방법으로 너비 우선 탐색을 실시하고, 경로가 존재하므로 종료하게 된다.

BF 알고리즘은 가장 빠른 시간 구간을 가진 노드부터 방문하기 때문에 1-4-7-8 경로를 따라 목적노드에 도달하게 되고 가장 빠른 시간은 1이 된다. 그 다음에는 1번 노드에서 2번 노드로 오는 링크의 가장 빠른 시작시간이 1이고, 1번 노드에서 3번 노드로 오는 링크의 가장 빠른 시작시간이 2이기 때문에, 더 이상 노드를 방문하지 않고 알고리즘을 종료하므로, 5번과 6번 노드는 방문하지 않게 된다.

2. BF 알고리즘

먼저 알고리즘을 기술하기 위해 필요한 용어들을 정의하기로 한다.

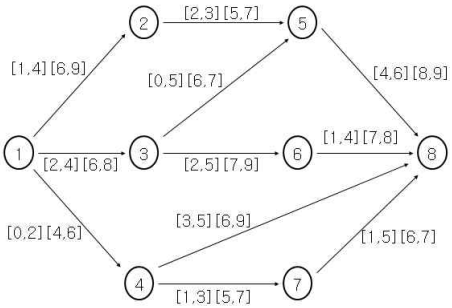

그림 1. 최초 구간 계산 문제의 예
Fig. 1. First Slot Computation Problem

정의 1 : D[i]

D[i]는 출발 노드 s에서 노드 i까지 데이터를 보낼 수 있는 모든 사용가능한 시간 구간들의 집합이다. 초기값으로 D[s] = {[0, ∞]}이다.

정의 2 : ∪ (합집합 연산)

두 개의 구간 리스트 A와 B가 있을 때 합집합(union) 연산 A∪B는 A의 구간들과 B의 구간들을 합친 집합이다.

정의 3 : ∩ (교집합 연산)

두 개의 구간 리스트 A와 B가 있을 때 교집합(intersection) 연산 A∩B는 A의 구간들과 B의 구간들 중에서 겹치는 구간들의 집합이다.

예를 들어, A = {[3, 7], [10, 15]}이고 B = {[4, 5], [9, 11], [14, 16]}이면, A∪B = {[3, 7], [9, 16]}, A∩B = {[4, 5], [10, 11], [14, 15]}가 된다.

정의 4 : NE 연산

노드 i와 노드 j사이에 에지 <i, j>가 존재하고 에지 <i, j>의 시간 구간들의 집합을 list(i, j)라 할 때 NE(i, j) = D[i] ∩ list(i, j)로 정의한다.

노드 j로 들어오는 에지가 <i, j> 하나이면 D[j] = NE(i, j)이다. 일반적으로 노드 j로 들어오는 에지가 여러 개일 때는 다음과 같은 식으로 구할 수 있다.

$$D[j] = D[j] \cup (\cup_{<i,j> \text{ in } E} NE(i, j)) \quad \cdots\cdots(1)$$

정의 5 : EST(earliest start time)

시간 구간들의 집합 S가 {[s_1, e_1], [s_2, e_2], …… , [s_m, e_m]}이고, 시간 t가 주어졌을 때 EST(S, t)는 S에 속한 시간 구간들 중에서 t보다 작지 않으면서 가장 빠른 시간으로 정의한다.

```
Algorithm BF(source, destination)
{
    initializePQ();
    insertPQ(source, 0);
    earliest = ∞;
    D[source] = {[0, ∞]}
    curTime = 0;
    while (PQ is not empty) {
       (p, pEST) = deleteminPQ();
       if (pEST < earliest) {
          curTime = EST(D[p], curTime);
          for (each s of <p,s> in E) {
             D'[s] = D[s] ∪ NE(p,s);
             if (D'[s] ≠ D[s]) {
                e = EST(D'[s], curTime);
                if (heapexist[s] = 0) insertPQ(s,e);
                else relocatePQ(s,e);
                D[s] = D'[s];
                if (s = destination) earliest = e;
             }
          }
       }
       else
          break;
    }
    return earliest;
}
```

그림 2. BF 알고리즘
Fig. 2. BF Algorithm

예를 들어 S = {[1, 3], [5, 8], [14, 15]}라 하면 EST(S, 0) = 1, EST(S, 2) = 2, EST(S, 4) = 5가 된다.

본 연구에서 제안하는 알고리즘은 노드 s에서 시작하여 s와 연결된 모든 노드 i에 대해 NE(s, i) 연산을 수행하여 D[i] 값들을 계산한다. 다음에는 모든 노드 중 EST가 최소인 노드를 선택한다. EST가 최소인 노드를 k라 하면 k와 연결된 모든 노드 i에 대해 NE(k, i) 연산을 수행하여 D[i] 값들을 계산한다. 이 과정을 목적노드에 도달할 때까지 반복한다. 남아있는 노드의 EST가 목적노드의 EST보다 크거나 같으면 종료한다. EST에 관한 연산은 우선순위 큐(priority queue)인 최소 힙(min heap)을 사용하는 데 다음 절에 상세히 기술하였다. 그림 2에 BF 알고리즘이 나타나 있는 데 사용된 프로시저(precedure)의 의미는 다음과 같다.

- initializePQ : 최소 힙의 초기화
- insertPQ : 원소를 최소 힙에 삽입
- deleteminPQ : 최소 힙에서 EST가 최소인 원소를 삭제
- relocatePQ : 최소 힙에서 원소의 위치를 변경

알고리즘에서 curTime은 초기에는 0값을 가지고 있다가 최소 힙에서 노드들이 선택되면서 그 값이 변하게 된다. 현재 최소 힙에서 삭제되는 노드를 i라 하면 노드 i의 EST 값을 curTime으로 정한다. EST 값이 작은 순으로 최소 힙에서 삭제되기 때문에 curTime은 목적노드에서 EST 값이 현재의 curTime보다 작을 수는 없게 된다. curTime은 앞으로 연산을 수행할 때 에지들의 시간 구간 리스트에서 curTime보다 작은 구간들을 제거하는 데 사용된다.

3. EST 저장과 관리

BF 알고리즘에서는 EST가 작은 노드부터 방문하여 NE 연산을 수행하므로 EST 값들은 최소 힙에 저장하면 된다. 최소 힙은 삽입(insert)과 최소 원소 삭제(deletemin) 연산들을 효율적으로 수행할 수 있는 우선순위 큐이다 [20].

노드 k의 선행노드가 하나가 아닐 경우에는 노드 k가 이미 최소 힙에 삽입되어 있는 데, 다른 선행 노드 j와 에지 <j, k>의 NE연산 결과에 의해 노드 k의 EST의 값이 변경될 수가 있다. 그림 3에 그 예가 나타나 있다.

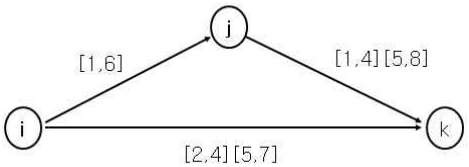

그림 3. EST가 변경되는 예
Fig. 3. Example of EST change

그림 3에서 D[i] = {[0, 10]}이라 가정하고, 노드 i와 노드 j 사이에 NE 연산을 수행하면 D[j] = {[1, 6]}이 되고 노드 i와

노드 k사이에 NE 연산을 수행하면 D[k] = {[2, 4], [5, 7]}이 된다. 이 때 노드 j(EST = 1)와 노드 k(EST = 2)가 최소 힙에 삽입된다. 그 다음 노드 j와 노드 k사이에 NE 연산을 수행하면 NE(j, k) = {[1, 4], [5, 6]}이 된다. (1) 식 에 따라 새로운 값을 구하면 D[k] = {[1, 4], [5, 7]}이 되고 EST 값이 2에서 1로 변하게 된다.

이 경우 최소 힙에서 노드 k의 위치를 새로운 EST 값에 따라 변경(relocation)시켜주는 것이 필요하게 된다. 그러나 최소 힙은 임의의 노드의 위치를 알 수 없고 최소 힙 내에서 노드의 위치를 변경하는 연산을 제공하지 않는다. 그러므로 노드의 위치를 변경하기 위해서는 노드가 최소 힙에 삽입될 때 삽입된 위치를 기억하고 있어야 한다. 일반적으로 힙은 일차원 배열로 구현되므로 노드가 삽입된 위치는 인덱스로 표시 가능하다. 본 연구에서는 노드들의 위치를 알기 위해 일차원 배열인 heapexist를 사용한다. heapexist[i] = 0이면 노드 i가 최소 힙 안에 존재하지 않음을 나타내고 heapexist[i] = x이면 노드 i가 최소 힙에서 x번째 원소에 입력되어 있음을 나타낸다. 그림 4에 그 예가 나타나 있다. 이 예에서 heapexist[2] = 3인데 노드 2가 일차원배열로 구현된 최소 힙에서 3번째 원소에 저장되었음을 나타낸다. 최소 힙에서 노드 밖의 번호는 일차원배열에서 원소의 번호를 나타내고 노드안의 쌍 (a, b)는 a는 노드의 번호, b는 노드 a의 EST 값을 나타낸다. 그림 4에서 노드 4의 EST 값이 18에서 13으로 바뀌었다면 노드 4와 그의 부모노드인 노드 5는 자리를 교환하게 되고 이동된 최종 위치를 heapexist에서 수정하게 된다. 즉, heapexist[4] = 2, heapexist[5] = 4가 된다.

노드 i의 EST가 감소하면 x번째 위치에서부터 노드 i를 삽입하는 연산을 수행하면 된다. 즉 노드 i의 EST를 x/2번째에 저장된 노드의 EST와 비교해서 작으면 두 노드의 위치를 교환하는 과정을 반복 수행한다.

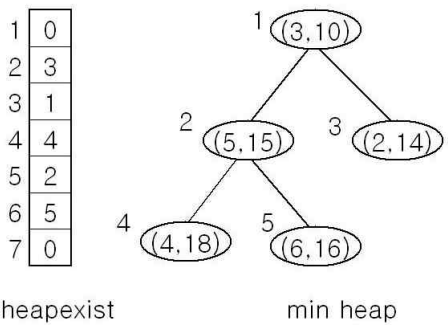

그림 4. 효율적인 EST 탐색과 갱신 자료구조
Fig. 4. Efficient Data Structure for EST Search and Update

IV. 성능 평가

제안된 휴리스틱들은 비쥬얼 C++ 6.0 언어를 사용해서 Intel(R) Core(TM) 2 CPU를 가진 PC에서 구현되고 실험되었다. 제안된 BF 알고리즘과 FS 알고리즘을 비교하기 위해 다양한 종류의 테스트 데이터를 임의(random)로 생성하였다.

생성된 네트워크에서 노드의 개수는 500, 1000, 1500, 2000 네 종류를 사용하였고, 각 노드마다 연결된 링크의 수는 3부터 6사이의 수를 사용하였으며 각 링크가 가지는 시간 구간의 개수는 10, 15, 20, 25를 사용하였다. 또, 네트워크상에서 출발노드에서 도착노드로 가는 경로들의 길이(path length)를 데이터 생성에 반영 도착노드로 먼저 출발 노드를 1이라 터 생목적 노드를 n이라 할h , 각 노드 i에서 에지를 생성할h 가있는 노드의 번호를 i - n/limit부터 i + n/limit으로 제한하였는데 limit 값이 작으면 노드로 먼저출발노드와 목적노드의 평균 경로의 길이가 짧아지게 되고, limit 값이 커지면 평균 경로의 길이가 길어지게 된다. Limit 값 먼는 25, 30, 35, 40을 사용하였다. 따라서 생성된 데이터의 경우의 수는 노드의 종류 4가지, 링크의 종류 4가지, 시간 구간의 종류 4가지, limit의 종류 4가지먼저총 256 경우이 생각 경우마다 10개의 데이터를 생성하여저 총 2560개의 데이터를 생성하였다. 실행시간은 시간 측정의 오차를 줄이기 위해 10번 실행시킨 후, 3평균 줄이사용하였다. 실험 결과에 의하면 2560개 데이터의 총 실행시간은 BF 알고리즘이 407.5초가 걸렸고 FS 알고리즘이 482.0초가 걸렸다. 2560개의 데이터 중 실행시간이 BF 알고리즘이 빠른 경우는 1944개였고, FS 알고리즘이 빠른 경우는 612개 였으며, 같은 경우는 4개였다.

해(solution)가 없는 292개의 데이터의 경우는 총 실행시간이 BF 알고리즘이 4.5초 걸린 반면 FS 알고리즘은 85.9초가 걸려 BF 알고리즘의 성능이 매우 우수하였다.

알고리즘의 성능은 limit 값, 노드의 수, 링크의 수, 시간 구간의 수에 따라 달라지기 때문에 다음에는 각각의 요소에 대한 성능을 비교하였다.

표 1. Limit 값에 대한 BF와 FS의 승패 비교
Table 1. Number of wins, ties and loses of BF over FS on various limit values

	win	tie	lose
L=25	548	1	91
L=30	495	2	143
L=35	457	1	182
L=40	444	0	196
합계	1944	4	612

표 2. 노드의 수에 대한 BF와 FS의 승패 비교
Table 2. Number of wins, ties and loses of BF over FS on the number of nodes

	win	tie	lose
N=500	311	4	325
N=1000	480	0	160
N=1500	552	0	88
N=2000	601	0	39
합계	1944	4	612

표 3. 링크의 수에 대한 BF와 FS의 승패 비교
Table 3. Number of wins, ties and loses of BF over FS on the number of links

	win	tie	lose
E=3	617	0	23
E=4	531	1	108
E=5	435	2	203
E=6	361	1	278
합계	1944	4	612

표 1에 각 limit 값에 대해 BF 알고리즘의 실행시간이 FS 알고리즘의 실행 시간보다 빠른 경우(win), 같은 경우(tie), 늦은 경우(lose)의 데이터의 수가 나타나 있다. 표 1에서 보는 바와 같이 실험에 사용된 모든 limit 값에 대하여 실행시간이 빠른 데이터의 수가 BF 알고리즘이 FS 알고리즘보다 더 많았는데, limit 값이 작을수록 실행시간이 빠른 데이터의 수가 더 증가 하였다.

표 2에는 네트워크에서 노드의 수에 대한 경우가 나타나 있는 데, N = 500일 때를 제외하고는 실행시간이 빠른 데이터의 수가 BF 알고리즘이 FS 알고리즘보다 더 많았다.

표 3에 링크의 수에 대한 경우가 나타나 있는 데, 실험에 사용된 모든 링크의 수에 대하여 실행시간이 빠른 데이터의 수가 BF 알고리즘이 더 많았다.

표 1~ 표 3에서 본 바와 같이 실행시간이 빠른 데이터의 수가 BF 알고리즘이 더 많은 이유는 BF 알고리즘은 일반적으로 어떤 노드에 있는 구간의 시작시간이 현재까지 찾은 가장 빠른 시작시간 보다 늦은 경우에는 이 노드를 방문하지 않는 가지치기(pruning)를 하는 데 비해, FS 알고리즘은 각 구간의 시작시간에 대해 너비 우선 탐색을 실시하기 때문에 네트워크상의 모든 노드를 여러 번 방문하기 때문이다.

그림 5에 각 limit 값에 대한 실행시간이 나타나 있는 데 FS알고리즘의 실행시간을 1이라 하였을 때 BF 알고리즘의 실행시간의 비율이 나타나 있다. BF 알고리즘은 limit 값이 작아질수록 성능이 점점 더 향상되는 것을 알 수 있는 데, limit 값이 30이하일 때는 FS 알고리즘보다 더 우수하고 35를 넘어가면 FS 알고리즘이 약간 더 우수한 것으로 나타났다. limit 값이 적으면 BF 알고리즘은 목적노드에 빨리 도달하게 되고, 현재까지 가장 빠른 시간을 구한 뒤 가지치기가 조기에 일어나기 때문에 실행시간이 빨라지나, FS 알고리즘은 네트워크상의 모든 노드를 여러 번 방문하기 때문에 limit 값의 영향을 받지 않는다.

다음에는 노드의 수에 대한 성능 비교가 그림 6에 나타나 있는 데 그림 6에서 보면 노드의 수가 1000 이상일 때 BF 알고리즘이 더 좋은 결과를 얻었는데 그 이유는 FS 알고리즘의 실행시간은 노드의 수에 비례하는 데 비해, BF 알고리즘은 노드의 수가 많으면 가치치기가 더 많아지기 때문이다.

각 노드에 연결된 링크의 수에 대한 성능 비교가 그림 7에 나타나 있다. 그림 7에서 보면 링크의 수가 5일 때는 성능이 비슷하였고, 5보다 작을 때는 BF 알고리즘이, 5보다 클 때는 FS 알고리즘이 우수하였다. 그 이유는 네트워크에 사이클이 있는 경우에는, BF 알고리즘은 방문한 노드를 다시 방문할 수 있으므로 링크의 수가 많으면 실행시간이 더 걸리게 된다. FS 알고리즘도 너비 우선 탐색을 여러 번 실행하기 때문에 노드를 여러 번 방문하나, BF 알고리즘이 오버헤드가 약간 더 많은 것으로 관측된다.

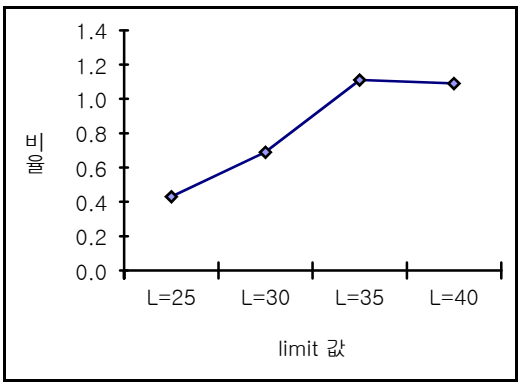

그림 5. Limit 값에 대한 실행 시간 비율
Fig. 5. Ratio of execution times on limit values

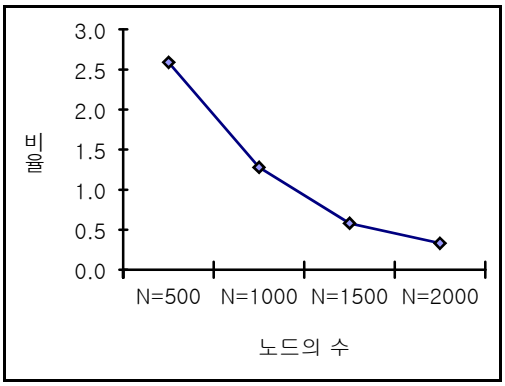

그림 6. 네트워크 크기에 대한 실행 시간 비율
Fig. 6. Ratio of execution times on the number of nodes

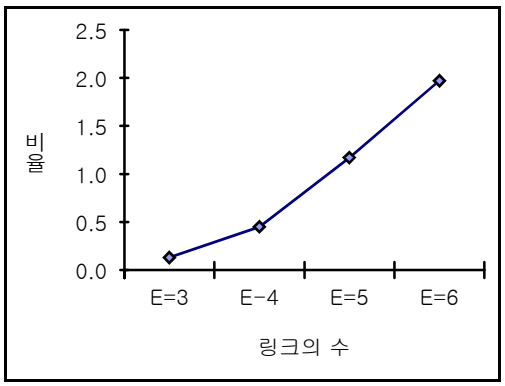

그림 7. 링크의 수에 대한 실행 시간 비율
Fig. 7. Ratio of execution times on the number of links

V. 결 론

e-Science와 상업적 분야에서의 다양한 응용들은 대규모 데이터 전송이 필요한데 이를 위해서는 스케줄 가능한 높은 대역폭과 낮은 대기 시간이 요구된다. 현재의 공용 인터넷에서는 예측 가능하고 효율적인 서비스를 제공하기 어렵기 때문에 동적으로 안정된 대역폭을 제공하기 위해서는 데이터 제공자와 사용자간에 전용 연결이 필요하게 된다. 이러한 네트워크들의 관리시스템은 대역폭 스케줄러가 탑재되어 있는 데, 스케줄러는 네트워크 형태 정보와 링크의 대역폭 할당 정보를 토대로 하여 사용자 요구에 의해 경로를 계산한다.

본 논문에서는 출발지와 목적지가 주어지고 최소 대역폭과 필요한 전송 시간이 주어졌을 때 전송 가능한 가장 빠른 시간을 찾는 알고리즘을 개발하고 기존의 알고리즘과 비교하였다.

실험 결과에 의하면 제안된 알고리즘이 기존의 알고리즘에 비해 실행시간이 빠른 데이터의 수가 훨씬 더 많았으며, 제안된 알고리즘은 링크의 수가 적고 노드의 수가 많은 대형 네트워크에서 대역폭 스케줄러에 탑재되어 좋은 성능을 나타낼 것으로 기대된다.

참고문헌

[1] J. Bunn and H. Newmann, "Data-Intensive Grids for High-Energy Physics," Grid Computing, John Wiley & Sons, 2003

[2] S. Sahni, N. Rao, S. Ranka, Y. Li, E. Jung, and N. Kamath., "Bandwidth Scheduling and Path Computation Algorithms for Connection Oriented Networks," International Conference on Networking, 2007

[3] E. Jung, Y. Li, S. Ranka, and S. Sahni, "An evaluation of in-advance bandwidth scheduling algorithms for connection-oriented networks," connrnational Symp. on Parallne Archinnctures, Algorithms, and Networks (ISPAN), pp. 133-138, 2008

[4] Dynamic resource allocation via GMPLS optical networks. *http://dragon.maxgigapop.net.*

[5] On-demand secure circuits and advance reservation system. *http://www.es.net/oscars.*

[6] X. Zheng and M. Veeraraghavan and N. S. V. Rao and Q. Wu and M. Zhu, " CHEETAH : Circuit-switched

high-speed end-to-end transport architecture testbed," IEEE Communications Magazine, 2005.
[7] N. S. V. Rao and W. R. Wing and S. M. Carter and Q. Wu, "Ultrascience net : Network testbed for large-scale science applications," IEEE Communications Magazine, 2005, expanded version available at *www.csm.ornl.go/ultranet.*
[8] Enlightened Computing, *http://www.enlightenedcomputing.org*
[9] Geant2, *http://www.geant2.net.*
[10] JGN II : Advanced Network Testbed for Research and Development, *http://www.jgn.nict.go.jp.*
[11] LHCNet : Transatlantic Networking for the LHC and the U.S. HEP Community, *http://lhcnet.caltech.edu/.*
[12] Hybrid Optical and Packet Infrastructure, *http://networks.internet2.eud/hopi.*
[13] Internet2. *http://www.internet2.edu.*
[14] User Controlled LightPath Provisioning, *http://phi.badlab.crc.ca/uclp.*
[15] D. Ghosal and B. Mukherjee N. S. V. Rao and Q. Wu and S. M. Carter and W.R. Wing and A. Banerjee, "Control plane for advance bandwidth Scheduling in ultra high-speed networks," In INFOCOM 2006 Workshop on Terabits Networks, 2006.
[16] Y. Li, S. Ranka, and S. Sahni, "In-advance path reservation for file transfers in e-science applications," IEEE Symposium on Computers and Communications, pp. 176-181, 2009
[17] Kannan Rajah, Sanjay Ranka and Ye Xia, "Scheduling Bulk File Transfer with Start and End Times," Computer Networks, Vol. 52(5), pp. 1105-1122, April 2008
[18] Kannan Rajah, Sanjay Ranka, Ye Xia, "Advance Reservation and Scheduling for Bulk Transfers in Research Networks," IEEE Transactions on Parallel and Distributed Systems, Vol. 20, No. 11, pp.1682-1697, November 2009.
[19] S. W. Lee and H. S. Song, "Characteristic of active optical ring network and performance evaluation in bandwidth on demand," Korea Society of Computer Information, Vol. 10, No. 6, December 2005
[20] E. Horowitz, S. Sahni, and D. Metha, "Fundamentals of Data Structures in C++," Compute Science Press, 1995

저 자 소 개

정 균 락

1980년 2월 : 한국과학기술원 전자계산학석사
1991년 2월 : 미네소타대학교 컴퓨터공학박사
1991년 ~ 현재 : 홍익대학교 컴퓨터공학과 교수
관심분야 : VLSI 알고리즘, 네트워크 알고리즘