論文 96-21-9-7

호스트 그루핑을 이용하는 연결 캐싱 기법

正會員 양 수 미*, 조 유 근*

A Connection Caching Technique using Host Grouping

Soomi Yang*, Yookun Cho* *Regular Members*

요 약

연결 지향 프로토콜을 쓰는 분산 환경에서 잦은 연결 설정으로 인한 오버헤드를 줄이기 위해 연결 캐싱 기법이 널리 사용되고 있다. 본 논문에서는 보다 효율적인 연결 캐싱 기법으로 전체 시스템을 통신이 많은 호스트들끼리 모아서 분리된 여러 개의 호스트 그룹을 만들고 동일 그룹내의 호스트 간에 우선적으로 캐쉬를 유지하는 기법을 제안하고 있다. 즉, 같은 그룹내에 있는 호스트끼리는 연결 사용 정보를 공유, 보다 오랜 시간동안 캐쉬에 머무르도록 하고 있다. 그리고 이 기법의 타당성을 검증하기 위해 평균 서비스 요청 시간 간격 (service request interarrival time), 평균 서비스 시간 (service time), 그룹 크기, 그루핑 형태, 그룹 특성 등의 측면에서 분석한 성능평가를 제시한다. 시뮬레이션 결과에 따르면 호스트 그루핑이 연결 캐싱의 모든 분석 척도에 대해 유효하며, 참조 패턴에 따른 적절한 그루핑이 더욱 성능을 높이는 것으로 나타났다. 또한 제안된 기법의 Markov 프로세스 모델을 제시, 분석하여, 시뮬레이션과 일관된 결과를 보이고 있다.

ABSTRACT

Connection caching technique is widely used to reduce the overhead incurred by frequent connection establishment in distributed computing environment using connection oriented protocol. We present an efficient connection caching scheme where we divide the system into several host groups and connection between the hosts in the same group is kept prior to others. In other words it makes the connection kept longer by sharing the connection use information between hosts in the same group. Every host group consists of hosts which have heavy intercommunication. And we present performance evaluation of the effect of host grouping upon connection caching and performance comparison of various grouping in several aspects including mean service request interarrival time, mean service time, group size, grouping and type of group. Simulation results show that host grouping is effective in every performance criterion and proper grouping of hosts enhances the performance. We also present analysis results for Markov process model of our scheme which are consistent with the simulation results.

*서울대학교 컴퓨터공학과
Department of Computer Engineering, Seoul National University
論文番號:95353-1011
接受日字:1995年 10月 11日

Ⅰ. 서 론

다수의 지리적으로 분산된 호스트가 참여하는 분산 계산 환경에서 사용되는 연결 지향(connection oriented) 프로토콜의 경우, 연결의 설정에 드는 오버헤드가 매우 커서, 가능한한 빈번한 연결의 설정과 해제를 줄이고, 이미 설정된 연결을 최대한으로 이용하고자 노력하고 있다[1, 3, 6, 7]. 즉, 연결 설정의 오버헤드를 줄이기 위해서, 연결을 캐쉬해 두거나[6, 7], 많이 쓰이리라고 예상되는 연결을 아예 미리 설정해두기도 한다. 그러나, 연결 캐쉉 기법은 광범위한 사용에도 불구하고, 성능 개선에 대한 정량적 정당성이 없이 직관에 의해 행해지는 경우가 많다[1, 4]. 그래서 연결 캐쉉 전략에 관련되는 파라미터들이 정확한 계량없이 역시 직관에 의해 결정되고 있으며, 캐쉉의 방법에 대한 연구도 미진한 상태다.

연결의 캐쉉은 연결 지향 프로토콜이나 데이터그램 (connectionless) 프로토콜에서 다 적용할 수 있겠으나, 연결 설정에 드는 오버헤드가 큰 연결 지향 프로토콜에서 보다 더 큰 성능의 향상을 얻을 수 있으며, 데이터그램 프로토콜에서는 캐쉬큐의 관리가 오히려 성능의 저하를 가져올 수 있으므로 본 논문은 연결 지향 프로토콜에서의 연결 캐쉉만을 대상으로 한다. 이러한 연결 캐쉉은 네트워크의 여러 계층 구조상에서 이루어 질 수 있으나, 본 논문에서는 응용층에서의 연결 캐쉉 기법을 제안하며, 그의 정량적 분석을 제시한다. 제안된 연결 캐쉉 기법에서는 대상으로 하는 전체 네트워크 시스템을 상호 연관된 호스트들로 이루어진 여러 개의 분리된 그룹으로 나누고, 그것이 연결 캐쉉에 미치는 영향을 조사한다. 이와 같이 특성 호스트 그룹이 참조 지역성을 가짐은 LAN에 대해서는 [10]에서, WAN에 대해서는 [11]에서 고고된 바 있다.

일반적으로 연결 지향 프로토콜은 신뢰성있는 데이터 전송을 보장받고자 하는 경우에 사용된다. 또한 QoS(Quality of Service) 를 만족시켜야하는 분산 멀티미디어 응용[15]이나 분산 실시간 응용[16]에서 그의 사용이 증가되고 있다. 그러한 환경에서, 다수의 원격작업이 있어 참조 패턴의 지역성이 존재한다면 본 논문에서 제시된 연결 캐쉉 기법이 효율적으로 사용될 수 있다. 또한 실시간 통신에서 admission control과 결합하여 성능향상을 도모할 수 있다[17]. 그리고 본 논문에서 신뢰성있는 연결을 가정하므로 현재의 기술 수준에서는, WAN보다는 LAN에서 더욱 적절하게 적용될 수 있다.

논문의 순서는, 우선 2장에서 관련 연구를 살펴보고, 3장에서는 논문에서 제시하는 호스트 그루핑을 감안한 연결 캐쉉의 기법을 설명한다. 4장에서는 시뮬레이션을 위한 기초 성능 분석과 시뮬레이션의 검증을 위한 분석을 한다. 5장에서는 시뮬레이션 결과가 제시된다.

Ⅱ. 관련 연구

연결 캐쉉은 버퍼 캐쉬 관리시 버퍼 캐쉬 블록을 일정 시간 유지 후에 반납하는 방법과 유사하다[5]. 만약에 참조 지역성을 보이는 많은 원격 작업이 존재하면, 그 작업에 사용되는 연결을 캐쉉한다. 이것은 연결 재설정에 따른 지연 시간을 없앤다. 즉, 시스템에서 연결의 해제를 바로 실행하지 않고, 연결을 유지, 다른 응용에서 그 연결을 재사용할 수 있도록 한다. 여기서 연결의 해제란 더 이상 그 연결을 통해 데이터가 전송되지 못하도록 하고, 관련 정보를 삭제하는 것이다. 예를 들면 TCP(Transmission Control Protocol)의 경우, 응용으로부터 close를 요청받은 후, 2MSL 상태로 들어가 더 이상 해당 연결을 사용할 수 없게 만드는 것을 말한다. 그러므로 아직 데이터의 전송이 가능한 half-close 상태는 연결이 해제되었다고 말할 수 없다[3, 9]. 그러므로 연결의 캐쉉은 운영체제에서 작업 종료에 따른 연결 해제 작업을 실시하지 않고 해당 연결정보(자료구조)를 보관하는 것으로, 이 시점부터 연결은 캐쉬 상태로 들어가며 연결의 보유가 시작된다. 연결의 보유는 연결이 재사용되거나, 시스템에 의해 강제로 해제될 때까지 계속된다. 강제 해제 정책은 여러 가지 방법으로 실시될 수 있다. 또한 휴면(idle) 상태로 유지되는 연결은 하드웨어(라인 bandwidth)와 시스템공간(자료구조)이 사용가능할 경우 추가의 비용을 거의 소비하지 않는다. 즉, TCP 프로토콜의 경우 keep alive timer를 동작시키지 않으면 휴면 상태의 연결에 대한 유지 비용은 없으며, keep alive timer를 동작시키더라도 2 시간 간격으로 동작되므로 연결 설정 요구가 빈번한 환경에서 별

로 부담이 되지는 않는다[9].

ISO 8473 addendum 3에서는 OSI(Open Systems Interconnection) 계층구조 상에서 네트워크 층의 SNDCF (Subnetwork dependent convergence function)의 subnetwork access function으로, 가상 연결의 경우 연결을 캐싱하도록 권고하고 있다[6]. [6]에서 권고된 방법은, 사용자가 원격 작업을 종료한 후, 일정 시간의 타임아웃 시간동안 연결이 재사용되기를 기다린 후, 그 기간내에 연결이 재사용되지 않을 경우, 연결을 해제하도록 하는 것이다. 타임아웃 시간을 결정하기 위한 정책은 권고하지 않고 있는데, 이에 관하여 [7]에서는 트래픽에 따른 가변 타임아웃 정책을 제시하고 있다. 그러나 저자는 연결 캐싱이 가지는 상호 효과를 무시하고 단지 한 호스트 내에서의 통계치만을 다루고 있으며, 성능을 최적화하기 위해서는 연결에 참여하고 있는 두 호스트가 상호 협조하여야 함을 결론에서 지적하고 있다.

본 연구에서는 연결 양끝의 호스트가 서로 상호 작용하며 연관되는 점을 감안한 연결 캐싱을 제시하고 있다. 이러한 연결 캐싱 정책에 대한 기초적인 정량적 분석은 [4]에 제시되어 있다. [4]에서는, 다양한 캐쉬큐 크기와 참조 패턴, 평균 서비스 시간 간격, 평균 서비스 시간, 캐쉬 블록 대체 정책 등에 관한 연결 캐싱의 성능분석이 이루어졌다. 본 논문은 그 결과를 바탕으로 하면서, 호스트 그루핑에 대한 분석을 추가하고 있다. 또한 [8]에서는 본 논문에서 제안한 기법을 적용한 효율적인 분산 시스템을 소개하고 있는데, 연결의 캐싱과 더불어 효율적인 데이터 전송방식, 분산 프로그래밍 언어 등이 제안되어 있다.

Ⅲ. 호스트 그루핑을 감안한 연결 캐싱

본 논문에서는 연결 캐싱의 전략으로, 서로 통신이 많은 호스트들을 묶어 그룹을 형성한다. 그리고 동일 그룹 내에서 연결되어 있는 호스트들은 서로 협동적으로 연결을 캐쉬한다. 호스트 A가 호스트 B로부터 원격 서비스를 받고자 하여 연결 x를 설정해야 될 경우에, 호스트 그루핑을 감안한 연결 캐싱은 그림 1과 같은 절차로 진행된다. 이 절차는 호스트 A가 수행하는 작업을 나타낸다. 호스트 B는, 호스트 A와 같은 그룹에 속할 경우, 단계 9가 수행될 때 x를 캐쉬하여

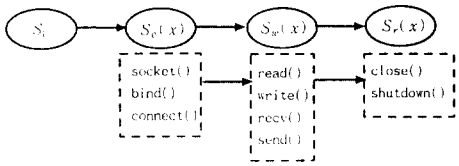
```
1. request_connection(x);
2. t = search_CACHE(x);
       /* t=i if CACHE(i) has x,
          else t=j if CACHE(j) has {y| y는 최장 보유시간의 연결}
3. if (t == i)
4.     reuse x;
5. else
6.     release(y);
7.     connection_establish(x);
8.     work with x;
9. cache_connection(x);
10. if (rereferenced (x))
11.     goto 1;
12. if (timeout reached)
13.     release(x);
```

그림 1 연결 캐싱 절차

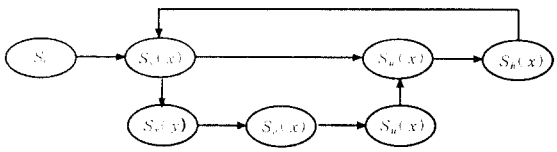
양방향 캐싱이 되도록 한다.

단계 1의, 연결 x를 설정하라는 요청은 구체적으로 호스트 이름 B와 해당 서버에 관련된 포트 번호를 가지고 이루어진다. 단계 2에서는 캐쉬 표의 탐색이 수행된다. 캐쉬 표의 탐색결과로, 만약에 요청된 연결 x가 캐쉬되어 있으면 해당 연결정보(예를 들어 소켓 디스크립터)에 대한 인덱스를 돌려 주며, 연결 x가 캐쉬되어 있지 않을 경우 가장 보유기간이 긴 연결에 대한 정보를 돌려준다. 연결이 캐쉬되어 있을 경우 단계 5-8의 과정이 생략된다.

그림 1의 연결 캐싱 절차에 따른 상태 전이를 그림 2에 보인다. 연결 설정에 소요되는 시간을 E, 순수 작업 시간을 W, 연결 해제 시간을 R, 캐쉬 표 탐색 시간을 S, 캐쉬 블록 대체 시간을 U, 연결 보유 시간을 H라 한다. 캐쉬 블록 대체 시간은 캐쉬 블록의 정보를 갱신하는 데 드는 시간이다. 또한 E의 시간을 소비하는 상태를 S_e, W의 시간을 소비하는 상태를 S_w, R의 시간을 소비하는 상태를 S_r, S의 시간을 소비하는 상태를 S_s, U의 시간을 소비하는 상태를 S_u, H의 시간을 소비하는 상태를 S_h, 그 이외의 상태를 S_0로 한다. 그러면 연결 캐싱을 하지 않는 경우의 상태 전이는 그림 2(a)와 같고, 연결 캐싱을 할 경우의 상태 전이는 그림 2(b)와 같다. 각 상태의 표시는 그림 1의 연결 캐싱 절차에 따라, 해당 연결을 괄호안에 포함하고 있다. 그리고 Berkeley Socket의 설정, 사용 과정[3]과의 비교를 socket system call을 사용하여 그림 2 (a)에 삽



(a) 연결 캐싱을 하지 않을 경우의 상태 전이도



(b) 연결 캐싱을 할 경우의 상태 전이도

그림 2 상태 전이도

입하였다.

연결 캐싱을 위해, 각 호스트는 연결 캐쉬큐를 유지한다. 캐쉬되는 각 연결 정보는 각기 한 단위의 연결 캐쉬큐 자료구조를 차지하는데, 그것을 캐쉬 블록이라고 부른다. 캐쉬 블록은 호스트 정보와 서버에 관한 정보(예를 들어 포트 번호)와 연결 정보(예를 들어 소켓 디스크립터)를 포함한다. 또한 H를 할당할때는, 타임아웃 기법을 병행적으로 적용한다. 그러나 타임아웃 시간을 충분히 길게 잡아서 연결 캐싱의 효과를 감소시키지 않도록 한다.

캐쉬 블록중 대체될 블록은 보유기간이 가장 긴 것을 선택하며, 이를 LHT (Longest Holding Time) 캐쉬 블록 대체 정책이라 부른다. LHT 캐쉬 블록 대체 정책을 구현하기 위해서, 각 캐쉬 블록은 타임스탬프를 포함한다.

연결 캐싱은 두 가지 형태로 구분된다. 첫 번째는 단방향 캐싱으로 연결 양 끝의 호스트가 다른 그룹에 속해 있는 경우에 해당한다. 이 경우 송신 호스트는 해당 연결 정보를 캐쉬하지만 수신 호스트는 캐쉬하지 않는다. 여기에서 송신 호스트는 연결의 요청을 보낸 호스트이고, 수신 호스트는 연결 요청을 받은 호스트이다. 연결 요청의 송수신 여부는 데이터의 전송 방향과는 무관하다. 두 번째는 양방향 캐싱으로 연결 양 끝의 호스트가 같은 그룹에 속해 있는 경우에 해당한다. 이 경우 송수신 호스트 모두가 해당 연결 정보를 캐쉬한다.

그룹의 특성은 부하 특성에 따라 두 가지로 분류한다. 하나는 같은 부하 특성(부하 지표)을 가지는 동종의 호스트들로 이루어진 균형 그룹이고, 다른 하나는 상이한 부하 특성(부하 지표)을 가지는 이종의 호스트들로 이루어진 불균형 그룹이다.

본 논문에서는 단방향 캐싱, 양향 캐싱, 균형 그룹, 불균형 그룹의 각 경우에 대하여 성능 향상에 미치는 영향을 알아보고 그에 따른 그루핑의 효과를 분석한다.

Ⅳ. 연결 캐싱의 성능

4. 1 기초 성능 분석

연결 캐싱에 관련된 한정된 네트워크 시스템이 모두 m개의 호스트를 포함하고 있고, 한정된 관찰 기간 동안의 총 연결 요청 즉, 서비스의 갯수를 n이라 하자. 그리고 C_{ij}와 D_{ij}는 호스트 i가 호스트 j로, 그림 1의 연결 캐싱 절차를 1회 실행하는데 드는 시간으로, 각각 연결 캐싱이 적용되지 않을 경우와 연결 캐싱이 적용될 경우를 나타낸다고 하자. 그러면,

$$C_{ij}=\sum_{k=1}^{n} F_k+\sum_{k=1}^{n} R_k,$$

$$D_{ij}=\sum_{k=1}^{n} F'_k+\sum_{k=1}^{n} H_k+\sum_{k=1}^{n} R_k \qquad (1)$$

이고, 여기서 k는 k번째 서비스를 가리킨다. 그리고 F_k는 연결 캐싱을 하지 않는 경우에, k번째 서비스를 받기 위하여 연결을 시도하는 시점으로부터 작업 완료(서비스 수행 완료)까지 드는 시간이고, F'_k는 연결 캐싱을 하는 경우에 작업 완료까지 드는 시간으로 다음과 같다.

$$F_k=E_k+W_k,$$
$$F'_k=S_k+E_k+U_k+W_k,\ 1\le k\le n \qquad (2)$$

이다. (1)-(2)에서 R_k, H_k, E_k, W_k, S_k, U_k는 서비스 k에 대하여, 그림 2의 상태 전이도에서와 같은 의미로 사용되었다. 그리고 U_k와 E_k는 연결이 캐쉬되어 있을 경우 0이고, 연결이 재사용될 경우 R_k가 0이다.

또한 각 호스트는 일정량의 부하를 가지게 되는데,

시뮬레이션에서 다음 두 가지를 부하 지표로 한다.

- I_a: 평균 서비스 요청 시간 간격
- U_s: 평균 서비스 시간

서비스 요청 시간 간격은 부하 발생 빈도를 나타내며, 서비스 시간은 서비스 요구를 처리하는데 소요되는 순수 작업 시간을 나타낸다. 두 지표는 지수 분포를 따르는 무작위 변수로 모델링되었다.

연결 캐쉬에 관련된 오버헤드는 앞의 (1)-(2) 에서 E, S, U, R 등이다. O를 연결 캐쉬을 하지 않을 경우의 오버헤드, O'를 연결 캐쉬을 하는 경우의 오버헤드라고 할때, 연결 캐쉬을 하지 않을 경우의 그룹 전체의 오버헤드 O_g와 연결 캐쉬을 하는 경우의 그룹 전체의 오버헤드 O_g'로부터 O와 O' 각각을 구해보자. 전체적으로 n개의 서비스 요구가 발생되고 히트율(hit ratio)이 $h(0<=h<=1)$일 경우, (1)과 (2)로부터 O_g는 n번의 E와 n번의 R을 포함한다.

$$\begin{aligned} O_g &= \sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{n}(C_{ij}-W_k)=\sum_{k=1}^{n}(F_k-W_k+R_k) \\ &= n*(E+R) \\ O &= E+R \end{aligned} \tag{3}$$

그리고 O_g'는 대략 $n*(1-h)$번의 E, U, R, 그리고 n번의 S를 포함한다.

$$\begin{aligned} O_g' &= \sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{n}(D_{ij}-W_k-H_k-R_k) \\ &+\sum_{l=1}^{n_2}R_l=\sum_{k=1}^{n}(F_g'-W_k)+\sum_{l=1}^{n_2}R_l \\ &\approx n*(1-h)\times(E+U+R)+n*S \\ O' &\approx (1-h)*(E+U+R)+S \end{aligned} \tag{4}$$

위에서 n_2는 재사용되지 않고 해제되는 연결의 갯수이다.

성능 향상의 측도로서 다음의 네 가지 성능 지표를 측정한다. 여기서 n_1은 한 호스트에 대하여 종료된 서비스의 개수이다.

오버헤드 감소율:

- P: 한 호스트의 오버헤드 감소율

$$P=\frac{\sum_{i=1}^{n_1}O_i-\sum_{i=1}^{n_1}O_i'}{\sum_{i=1}^{n_1}O_i} \tag{5}$$

- P_g: 한 그룹의 오버헤드 감소율

$$P_g=\frac{\sum_{i=1}^{n}O_i-\sum_{i=1}^{n}O_i'}{\sum_{i=1}^{n}O_i} \tag{6}$$

전체 수행시간 감소율:

- Q: 한 호스트의 전체 수행 시간 감소율

$$Q=\frac{\sum_{i=1}^{n_1}(W_i+O_i)-\sum_{i=1}^{n_1}(W_i+O_i')}{\sum_{i=1}^{n_1}(W_i+O_i)} \tag{7}$$

- Q_g: 한 그룹의 전체 수행 시간 감소율

$$Q_g=\frac{\sum_{i=1}^{n}(W_i+O_i)-\sum_{i=1}^{n}(W_i+O_i')}{\sum_{i=1}^{n}(W_i+O_i)} \tag{8}$$

이러한 P, Q, P_g, Q_g의 값은 표 1부터 표 7까지 제시되어 있다.

4.2 분석 모델

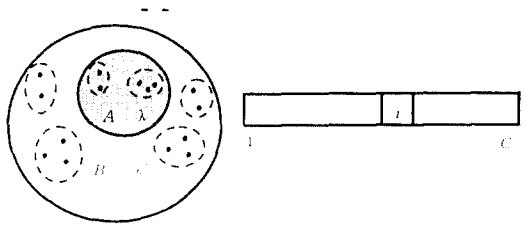
캐쉬 관리 전략인 LHT(Longest Holding Time) 캐쉬 블록 대체 정책을 분석하여, 적절한 호스트 그루핑이 성능의 향상을 가져올수 있슴을 보이고자 그림 3와 같은 분석 모델을 구성하였다. 이것은 한 수신 호스트(예를 들어 R)를 기준으로 하여 호스트 그루핑의 영향을 분석한 것이다. 그림 3 (a) 에서 연결 캐쉬의 대상이 되는 전체 공간을 크게 둘로 나누었는데, 회색 면의 공간이 서로 통신이 많은 호스트의 그룹. A이고 나머지 하얀 면의 공간이 호스트의 그룹 B이다. 각 그룹은 다음과 같이 호스트들을 포함하며, 그림에서 점선 경계의 타원으로 표시되었다. 즉, 그룹 A가

N_A개의 호스트를 포함하고, 그룹 B가 N_B개의 호스트를 포함할 경우,

$$A=\bigcup_{i=1}^{N_A} A_i,\quad B=\bigcup_{i=1}^{N_B} B_i$$

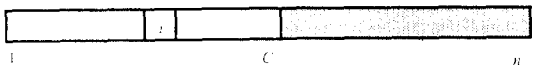
이다. 또한 각 호스트는 자신을 송신 호스트로 하는 연결들을 포함하는데, 각각의 연결은 점선으로 된 타원 내부에 점으로 표시되었다. 즉, 각각의 연결은 다음과 같이 각 호스트 그룹의 요소로 포함된다.

정의: 호스트 i와 호스트 j를 연결하는 연결 x_{ij}는 다음과 같이 그룹 G에 속한다.
$x_{ij}\in G$ if $i\in G$, $j\equiv R$ for $G=\{A, B\}$, $R\in A$



(a) 그룹 공간과 전체 공간

(b) 캐쉬큐의 상태



(c) 연결큐

그림 3 LHT 분석 모델

그러므로 각 호스트 그룹은 연결의 그룹으로 생각할 수 있으며, 이하의 분석에서 A와 B 각각은 연결의 집합을 가리키는 것으로 간주한다.

A에서의 서비스 요청 빈도를 λ, B에서의 서비스 요청 빈도는 λ' 라고 하자. 시스템의 상태는 그림 3 (b)에 보인 바와 같이 캐쉬큐의 상태로 표현하며, C개의 캐쉬 블록이 연결의 보유 시작 시간 순서로 배열되어 있다. 이러한 캐쉬큐의 형태를 모든 연결에 대한 것으로 확장하면 그림 3 (C)와 같이 된다. 즉 모든 연결(특정 수신 호스트에 관련된 모든 연결)이 보유시작 시간 순서(또는 서비스 종료 시간 순서, 일정한 서비스 시간 가정)로 순서화되어 있다고 생각할 수 있다(보유되지 않을 경우 $C+1$번째에 삽입). 이와 같이 캐쉬큐를 확장시킨 것을 연결큐라고 하자.

그러면, 연결에 대한 요청 열 $w=r_1 r_2\cdots r_t\cdots$에 대하여 연결큐의 시퀀스는 $s_0 s_1\cdots s_t$가 된다. 그러면 연결큐 s_t는 n-tuple $(j_1,\cdots, j_n)$이며, 이 때에 j_i는 시간 t에, 최근 i번째로 연결 보유가 시작된 연결이다. 그룹 A에 속한 특정 연결 (예를 들어 x)가 연결큐 상에서 시간의 경과에 따라 움직이는 것을 관찰해 보자. 무작위 시퀀스 $E_0 E_1 E_2\cdots E_t\cdots$를, 연결큐 s_t의 i번째 캐쉬 블록이 연결 x일 때 $E_t=i$가 되도록 정의한다. 그러면 모든 $t\geq 1$에 대하여 $1\leq E_t\leq n$이고, 이 시퀀스는 이산 매개변수, 이산 상태 stochastic 프로세스가 된다. 그림 4 (a)에 보인 연결큐의 갱신 절차에 의해, 연결큐 s_{t+1}에서의 연결 x의 위치는 다음 연결 요청 r_{t+1}과 연결큐 s_t에서의 연결 x의 위치에 의해 결정되며, 그 이전의 연결큐에서의 위치와는 무관하다. 그러므로 위의 열은 이산-매개변수 Markov 체인이 된다[12, 13, 14, 15].

그림 4 (a)에 보인 연결큐 갱신 절차를 관찰하여 호스트 그루핑을 하지 않을 경우, 체인의 전이 확률을 구한다. 그러면:

$$\begin{aligned}
p_{i1}&=P(E_{t+1}=1\,|\,E_t=i)\\
&=P(r_{t+1}=x)=\frac{\lambda}{A}*\frac{1}{\lambda+\lambda'},\quad 1\leq i\leq n,\\
p_{ii}&=P(E_{t+1}=i\,|\,E_t=i)\\
&=\left(\frac{A-1}{n-1}*\frac{\lambda}{A}+\frac{B}{n-1}*\frac{\lambda'}{B}\right)\\
&\quad *(i-1)*\frac{1}{\lambda+\lambda'},\quad 2\leq i\leq n,\\
p_{i,i+1}&=P(E_{t+1}=i+1\,|\,E_t=i)\\
&=\left(\frac{A-1}{n-1}*\frac{\lambda}{A}+\frac{B}{n-1}*\frac{\lambda'}{B}\right)*(n-i)\\
&\quad *\frac{1}{\lambda+\lambda'},\quad 1\leq i\leq n-1,\\
p_{ij}&=0, \qquad\qquad otherwise
\end{aligned}$$

위에서,

$$p_{ij}\geq 0,\ \sum_{j=1}^{n} P_{ij}=1,\ i,\ j=1,\ 2,\ldots,n$$

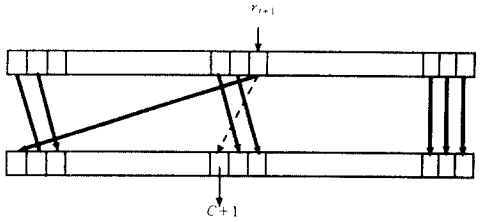
이므로 Markov 체인의 조건[14]을 만족함을 볼 수 있다. 상태도는 그림 4(b)에 주어진 바와 같다. (상태 1과 상태 $C+1$로 전이할 경우 서비스 상태를 거쳐야 하나, 그 영향이 미미하여 본 분석에서는 생략하였다.) 상태도에서 보는 바와 같이, 모든 i에 대해 $p_{i1}>0$임을 가정할 때, 체인이 aperiodic하고 irreducible하다. 그러면 평형-상태 확률 벡터 $\mathbf{v}=(v_1, v_2, \cdots, v_n)$ (각 위치에 있을 확률)가, Markov chain의 limiting probabilities에 관한 정리[14]에 의해, 다음과 같은 관련식들(system of equations)로 부터 구해진다:

$$v_1=\sum_{i=1}^{n} v_i * \frac{\lambda}{A} * \frac{1}{\lambda+\lambda'}$$

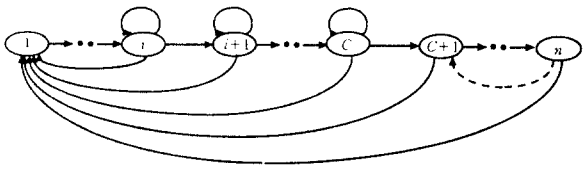
$$v_i=v_{i-1} * \left(\frac{A-1}{A} * \lambda+\lambda'\right) * \frac{n-i+1}{n-1} * \frac{1}{\lambda+\lambda'}$$

$$+ v_{i-1} * \left(\frac{A-1}{A} * \lambda+\lambda'\right) * \frac{i-1}{n-1} * \frac{1}{\lambda+\lambda'}, \quad 2 \le i \le n,$$

$$\sum_{i=1}^{n} v_i = 1.$$



(a) 캐쉬큐 갱신 절차



(b) 상태도

그림 4 연결 x에 대한 캐쉬큐 갱신 절차와 그에 따른 상태도

이 식들의 계산 결과를 그림 5에 보인다.

이번에는 그룹 A에 대해 호스트 그루핑을 할 경우, (그룹 A에 속한) 연결 x에 대한, 체인의 전이 확률을 구한다. 그룹 A에 속하지 않은 연결이 요청되었을 경우 해당 캐쉬 블록이 점선과 같이 $C+1$번째 위치로 이동하는 것으로 가정한다. 그러면:

$$p_{11}=P(E_{t+1}=1 \mid E_t=1)=P(r_{t+1}=x) \cup P(r_{t+1} \in B)$$
$$=\left(\frac{\lambda}{A}+\lambda'\right) * \frac{1}{\lambda+\lambda'},$$

$$p_{i1}=P(E_{t+1}=1 \mid E_t=i)=P(r_{t+1}=x)$$
$$=\frac{\lambda}{A} * \frac{1}{\lambda+\lambda'},$$

$$p_{ii}=P(E_{t+1}=i \mid E_t=i)$$
$$=\left[\frac{\lambda}{A} * (i-1)+\lambda'\right] * \frac{1}{\lambda+\lambda'}, \qquad 2 \le i \le C,$$

$$p_{i,i+1}=P(E_{t+1}=i+1 \mid E_t=i)$$
$$=\frac{\lambda}{A} * (A-i) * \frac{1}{\lambda+\lambda'}, \qquad 1 \le i \le C,$$

$$p_{ii}=P(E_{t+1}=i \mid E_t=i)$$
$$=\left[\left(\frac{A-C-1}{n-C-1} * \frac{\lambda}{A}+\frac{B}{n-C-1} * \frac{\lambda'}{B}\right)\right.$$
$$\left. * (i-C-1)+\frac{\lambda}{A} * C\right] * \frac{1}{\lambda+\lambda'}, \quad C+1 \le i \le n,$$

$$p_{i,i+1}=P(E_{t+1}=i+1 \mid E_t \mid E_t=i)$$
$$=\left(\frac{A-C-1}{n-C-1} * \frac{\lambda}{A}+\frac{B}{n-C-1} * \frac{\lambda'}{B}\right)$$
$$* (n-i) * \frac{1}{\lambda+\lambda'}, \quad C+1 \le i \le n-1,$$

$$p_{ij}=0, \qquad \textit{otherwise}$$

그러면 이전의 경우와 마찬가지로 다음의 관련식들로부터 평형 상태 확률 벡터 $\mathbf{v}$를 구할 수 있다.

$$v_1=v_1 * \frac{\lambda'}{\lambda+\lambda'}+\sum_{i=1}^{n} v_i * \frac{\lambda}{A} * \frac{1}{\lambda+\lambda'},$$

$$v_i=v_{i-1} * \frac{A-i+1}{A} * \frac{\lambda}{\lambda+\lambda'}$$
$$+ v_i * \left[\frac{\lambda}{A} * (i-1)+\lambda'\right] * \frac{1}{\lambda+\lambda'}, \qquad 2 \le i \le C,$$

$$v_{C+1}=v_C * \frac{A-C}{A} * \frac{\lambda}{\lambda+\lambda'}+v_{C+1} * \frac{C}{A} * \frac{\lambda}{\lambda+\lambda'},$$

$$v_i=v_{i-1} * \left(\frac{A-C-1}{A} * \lambda+\lambda'\right) * \frac{n-i+1}{n-C-1} * \frac{1}{\lambda+\lambda'}$$

$$+ v_i * \left[\left(\frac{A-C-1}{A} * \lambda+\lambda'\right) * \frac{i-C-1}{n-C-1}+\frac{\lambda}{A} * C\right] * \frac{1}{\lambda+\lambda'},$$
$$C+2 \le i \le n,$$

$$\sum_{i=1}^{n} v_i = 1.$$

이 식들의 계산 결과를 그림 5에 보인다.

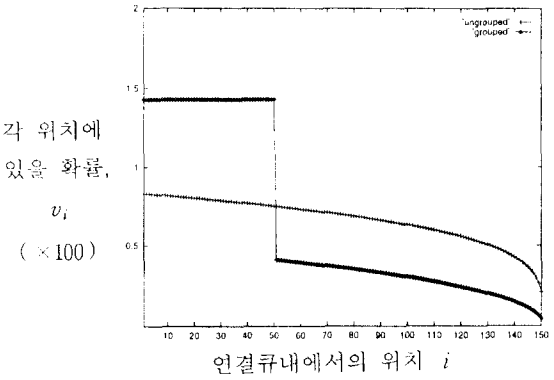


그림 5 특정 연결이 연결큐내에서 위치 i에 있을 확률
($A=70$, $N=150$, $C=50$, $\lambda=0.7$, $\lambda'=0.5$)

그리고 hit miss율 v_u는 다음과 같이 정의된다.

$$v_u = \sum_{i=C+1}^{n} v_i,$$

그러면 v_u는 연결이 캐쉬큐내에 없어서, hit miss되는 확률로, 이 값을 줄이는 것이 연결 캐싱의 목적이라고 할 수 있다. 그림 5에서 그루핑이 되지 않았을 경우 $v_u=0.60$, 그루핑이 되었을 경우 $v_u=0.29$로, 호스트 그루핑을 하는 것이 hit miss율을 낮추는 것을 알 수 있다.

A와 C의 변화에 따른 hit miss율 v_u의 변화와 그에 따른 성능지표 P, Q의 값의 변화를 (1)-(7)로부터 구하면 그림 6과 같다. E, R, U, S값은 시뮬레이션에서와 같은 값으로 하고 $I_a=1.0$이며, 히트율 $h=1-v_u$이다. (단일 호스트에 대해 분석하므로 양방향 캐싱에서 보다 성능이 약간 떨어진다.)

그림 6(a)에서, A가 그루핑되었을 때 hit miss율이 더 작으며, 캐쉬 크기 C가 A에 근접할 때 hit miss율이 보다 더 작아지는 것을 볼 수 있다. 그림 6 (b)와 (c)에서도 A가 그루핑되었을 때 오버헤드 감소율 P와 전체 수행시간 감소율 Q가 더 크며, 캐쉬 크기 C가 A에 근접할 때, 보다 더 큰 값을 보여 성능이 크게 향상됨을 알 수 있다.

V. 시뮬레이션

5.1 시뮬레이션 환경

본 논문에 제시된 연결 캐싱의 효율을 여러 측면에

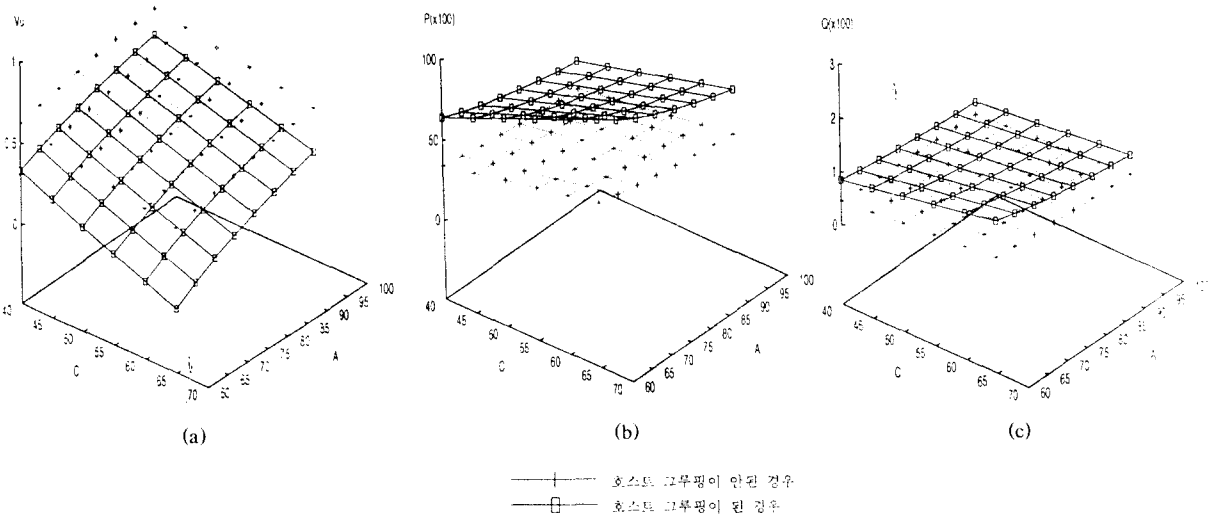


그림 6 A와 C의 변화에 따른 성능 비교
($N=150$, $\lambda=0.6$, $\lambda'=0.4$)

서 측정하기 위해 discrete-event 시뮬레이션을 수행하였다. 시뮬레이션 프로그램은 event-oriented simulation language인 smpl을 써서 작성하였다. 시뮬레이션의 입력 데이터인 평균 서비스 요청 시간 간격 I_a와 평균 서비스 시간 I_s는 지수 분포를 따른다. 또한 시뮬레이션 환경은 캐쉬큐의 크기, 참조 패턴, 연결 설정에 드는 시간(E), 연결 해제에 드는 시간(R), 캐쉬 블록 대체에 드는 시간(U), 표 탐색 시간(S) 등의 변수를 포함한다. 이 값들은 시스템 의존적인 변수이지만 이 값을 변수로 하는 것이 호스트 그루핑의 효과에 대한 분석에 큰 영향을 미치지 않으므로[4], 본 시뮬레이션에서는 SunOS 4.1 운영체제의 Sun4 머신과 2.6 MSD Mach 운영체제의 486 AT 머신을 TCP/IP 로 연결한 프로토타입 시스템을 만들고 그 시스템에서 측정된 값을 사용하였다. 즉, E를 0.4초, R를 0.002초, U를 0.01초, S를 0.007 초로 하였다. 시스템의 부하가 높아 측정 수치가 크지만, 상대적 크기만이 성능지표와 관계된다. 캐쉬큐의 크기는 50으로 하였다. 참조 패턴은 참조 지역성으로 대표하고, working set의 값을 다시 참조하는 확률로 표현, 80%로 하였다. 다양한 캐쉬큐 크기와 참조 패턴에 대한 분석은 [4]에 제시되어있다.

본 논문에서 제시되는 데이타는 약 10000 번의 단위 시뮬레이션의 실행 결과로부터 얻은 값을 평균한 것이며 특별히 언급되지 않은 경우 3 개의 호스트로 이루어진 그룹을 대상으로 하고, I_s는 30이며, LHT 캐쉬 블록 대체 정책을 적용한다. 그리고 모든 측정치는 성능의 향상율을 나타내는 백분율이다.

5.2 연결 캐쉬의 일반적인 성능

평균 서비스 요청 시간 간격 I_a와 평균 서비스 시간 I_s에 따른 연결 캐쉬의 일반적인 성능을 측정하고 그 결과를 표 1에 보인다. 표 1의 결과를 보면, 서비스 요청 간격이 길고 서비스 시간이 짧을수록 부하가 적고, 따라서 경쟁이 적으므로 더 나은 성능을 보인다. 그러나 성능이 포화점에 이른 유휴시스템에 대해서 더 이상의 성능 개선은 없다.

다른 연결 캐쉬 정책의 한가지인 timeout 기법[6,7]과의 비교를 하였다. 같은 부하지표에 대해, 캐쉬 크기에 제한을 두지 않고 timeout을 두어 캐쉬 관리를 하였다. 표 2는 timeout을 40.0으로 하였을 때 보이는 성능향상율이며, 소요되는 평균 캐쉬 크기를 추가로 기록하였다. 대체로 표 1의 결과보다 성능이 나쁘며, I_a가 1.0이고, I_s가 50.0일 때 성능이 좋아지나, 이때에 요구되는 캐쉬의 크기는 160% 정도 증가된 값이다.

표 2. 일반적 성능 비교 II(timeout 정책의 적용)

I_a	I_s								
	10.0			30.0			50.0		
	P_g	Q_g	Cache Size	P_g	Q_g	Cache Size	P_g	Q_g	Cache Size
1.0	67.90	2.64	50	66.47	0.89	70	65.99	0.53	80
2.0	68.45	2.66	45	68,00	0.91	60	66.98	0.54	65
3.0	68.62	2.67	45	68.17	0.91	50	67.78	0.55	55

5.3 그룹 크기

그룹 크기는 그룹에 있는 호스트의 갯수이다. 크기가 다른 그룹에 대한 연결 캐쉬의 영향을 보이기 위해 동일한 부하 특성을 가지는 다른 크기의 균형 그룹들에 대해 성능 평가를 하고 표 3에 그 결과를 보였다.

그룹내의 호스트의 갯수가 증가함에 따라 캐쉬 블록은 보다 빨리 소비되며, 그 결과 성능이 저하된다. 그러나 성능의 저하가 그룹 크기의 증가율에 비해 상대적으로 적은 양인 것을 볼 수 있다. 이러한 이득은 캐쉬 블록의 공유에서 오는 것으로, 이것으로부터 호스트 그루핑이 유용하리라는 것을 기대할 수 있다.

표 1. 일반적 성능 비교

I_a	I_s					
	10.0		30.0		50.0	
	P_g	Q_g	P_g	Q_g	P_g	Q_g
1.0	79.42	3.07	71.16	0.94	60.44	0.49
2.0	79.82	3.08	78.24	1.04	73.92	0.59
3.0	80.00	3.09	79.54	1.05	77.78	0.62

표 3. 그룹 크기에 따른 성능 비교

I_a	그룹 크기					
	1		3		5	
	P_g	Q_g	P_g	Q_g	P_g	Q_g
1.0	77.11	1.03	71.16	0.94	69.02	0.92
2.0	80.09	1.07	78.24	1.04	76.61	1.02
3.0	79.64	1.05	79.54	1.05	78.44	1.04

5.4 그루핑 방법

호스트 그루핑의 보다 나은 사용을 위해 여러가지 그루핑 방법을 시험해 보았다. 표 4는 네 개의 호스트 간에 많은 원격 작업이 있을 경우의 세 가지 그루핑 방법을 비교하고 있다. 첫 번째는 그루핑을 하지 않는 경우로 각 호스트는 단방향 캐쉬을 한다. 두 번째는 그룹 크기가 2 인 경우이다. 각 호스트는 자신이 속한 그룹내에서는 양방향 캐쉬을 하고, 그룹간에는 단방향 캐쉬을 한다. 세 번째는 그룹 크기가 4인 경우로 각 호스트가 양방향 캐쉬을 한다. 표에서 보는 바와 같이 각 경우에 대해 성능이 점차 증가한다. 이것은 참조 패턴에 따른 적절한 그루핑이 성능을 보다 더 향상시키며, 호스트간 통신이 많을 경우 호스트 그루핑을 하여 양방향 캐쉬을 하는 것이 더 유리함을 나타낸다.

표 4. 그루핑 형태에 따른 성능 비교

I_a	그루핑 형태					
	(0), (1), (2), (3)		(0, 1), (2, 3)		(0, 1, 2, 3)	
	P_g	Q_g	P_g	Q_g	P_g	Q_g
1.0	68,08	0.91	68.31	0.91	68.90	0.92
2.0	72.47	0.96	74.11	0.98	76.98	1.02
3.0	72.66	0.96	74.52	0.99	78.28	1.39

표 5에서는 여섯 개의 호스트로 이루어진 시스템에서 호스트 번호 0, 1, 2인 호스트들과 3, 4, 5인 호스트들 간에 원격 작업이 많이 있는 경우에 대해 성능 비교를 하였다. 서로 원격 작업이 많은 호스트끼리 그루핑이 되어 있는 경우 2-5% 정도 오버헤드가 더 감소하는 것을 볼 수 있다.

표 5. 그루핑 형태에 따른 성능 비교 II

I_a	그루핑 형태			
	(0, 1, 2), (3, 4, 5)		(0, 1), (2, 3), (4, 5)	
	P_g	Q_g	P_g	Q_g
1.0	68.32	0.91	66.28	0.89
2.0	75.86	1.01	72.14	0.96
3.0	77.31	1.03	72.87	0.97

5.5 평균 서비스 요청 시간 간격의 불균형

상이한 평균 서비스 요청 시간 간격의 불균형 그룹에 대하여 성능 비교를 하였다. 표 6 (a)는 호스트 그루핑을 하는 경우이고, 표 6 (b)는 호스트 그루핑을 하지 않는 경우이다. 각 표의 아래 부분에 주어진 균형 그룹에 대한 자료는 비교를 위해 첨가되었다. 두 표의 결과를 보면, 보다 짧은 서비스 요청 시간 간격을 가진 높은 부하의 호스트가 캐쉬 블록을 보다 빨리 잡음으로써 보다 나은 성능을 보이며, 호스트 그루핑을 하는 경우 3-4% 정도 오버헤드가 더 감소하는 것을 볼 수 있다.

표 6. 다른 서비스 요청 시간 간격을 가지는 그룹에 대한 성능 비교

(a) 호스트 그루핑을 하는 경우

I_a	호스트 1과 2에 대한 성능 향상율		호스트 1에 대한 성능 향상율		호스트 2에 대한 성능 향상율	
	P_g	Q_g	P	Q	P	Q
(1.0, 2.0)	75.89	1.00	74.78	0.99	78.02	1.04
(1.0, 3.0)	76.14	1.01	75.62	1.00	77.62	1.04
(2.0, 3.0)	79.28	1.05	79.52	1.06	78.94	1.05
(1.0, 1.0)	72.08	0.95	71.62	0.94	72.52	0.95
(2.0, 2.0)	78.49	1.03	78.84	1.04	78.17	1.02
(3.0, 3.0)	78.56	1.04	79.34	1.05	79.79	1.04

(b) 호스트 그루핑을 하지 않는 경우

I_a	호스트 1과 2에 대한 성능 향상율		호스트 1에 대한 성능 향상율		호스트 2에 대한 성능 향상율	
	P_g	Q_g	P	Q	P	Q
(1.0, 2.0)	72.51	0.97	73.95	0.99	69.61	0.93
(1.0, 3.0)	73.56	0.98	76.47	1.02	64.81	0.86
(2.0, 3.0)	75.28	1.00	76.95	1.02	72.81	0.98
(1.0, 1.0)	71.67	0.96	70.93	0.95	72.39	0.97
(2.0, 2.0)	74.80	1.00	75.07	1.00	74.53	1.01
(3.0, 3.0)	74.78	1.00	74.30	0.98	75.30	1.01

5.6 평균 서비스 시간의 불균형

상이한 서비스 시간을 가지는 불균형 그룹에 대한 성능 비교를 표 7에 보이고 있다. 표 7 (a)는 호스트 그루핑을 하는 경우이고, 표 7 (b)는 호스트 그루핑을 하지 않는 경우이다. 각 표의 아래 부분에 주어진 균형 그룹에 대한 자료는 비교를 위해 첨가되었다. 성능 평가 결과를 보면 긴 서비스 시간을 갖는 호스트가 조금 더 성능이 높아지는 것을 볼 수 있다. 이것은 한정된 캐쉬를 긴 서비스 시간의 호스트가 조금 더 많이 차지하고 있는 데서 오는 결과이다. 그러나 짧

은 서비스 시간대에서는 서비스 시간 차이에 따른 성능 차이는 거의 없다[4]. 표 7에서 호스트 그루핑을 하는 경우 2-4% 정도 오버헤드가 더 감소하는 것을 볼 수 있다.

표 7. 다른 서비스 시간을 가지는 그룹에 대한 성능 비교
(a) 호스트 그루핑을 하는 경우

I_s	호스트 1과 2에 대한 성능 향상율		호스트 1에 대한 성능 향상율		호스트 2에 대한 성능 향상율	
	P_g	Q_g	P	Q	P	Q
(10.0, 30.0)	79.48	1.54	79.08	3.04	79.86	1.04
(10.0, 50.0)	78.14	1.01	78.74	3.03	77.55	0.61
(30.0, 50.0)	76.58	0.75	7669	1.01	76.48	0.60
(10.0, 10.0)	80.11	3.08	80.07	3.11	80.15	3.06
(30.0, 30.0)	78.49	1.03	78.84	1.04	78.17	1.02
(50.0, 50.0)	74.47	0.59	74.15	0.59	74.78	0.59

(b) 호스트 그루핑을 하지 않는 경우

I_s	호스트 1과 2에 대한 성능 향상율		호스트 1에 대한 성능 향상율		호스트 2에 대한 성능 향상율	
	P_g	Q_g	P	Q	P	Q
(10.0, 30.0)	75.76	1.48	75.68	2.94	75.85	0.99
(10.0, 50.0)	74.59	0.98	74.91	2.91	74.26	0.59
(30.0, 50.0)	74.51	0.74	74.15	0.99	74.86	0.59
(10.0, 10.0)	75.77	2.92	75.43	2.93	76.11	2.90
(30.0, 30.0)	75.13	0.99	74.82	1.00	75.43	0.99
(50.0, 50.0)	72.59	0.58	71.75	0.58	73.43	0.58

Ⅵ. 결 론

'캐싱' 아이디어를 네트워크 연결에 대해 적용하며, 보다 효율적인 캐싱을 위해 호스트 그루핑을 도입하고, 그에 따라 기대되는 성능 향상에 대한 정량적 분석을 하였다. 원격 작업이 많고 또 참조 패턴이 지역성을 보이면, 네트워크 연결을 늦게 해제함으로써 여러 개의 연결을 캐쉬한다. 그리고 시스템내에서 통신이 많은 호스트들끼리 묶어서 분리된 호스트 그룹을 만들고 다른 호스트와의 연결에 우선하여 캐싱한다. 이러한 연결 캐싱 기법을 분석하여 성능 향상에 대한 정량적 수치를 얻었다. 연결 캐싱에 대한 분석은 평균 서비스 요청 시간 간격, 평균 서비스 시간, 그룹 크기, 그루핑 형태, 그룹 특성 등의 측면에서 이루어졌다.

시뮬레이션 결과에 따르면 연결 캐싱이 시스템의 성능을 높이며, 호스트 그루핑은 그러한 점을 보다 보강하며, 시스템 내에서의 호스트간의 연관성을 고려한 적절한 호스트 그루핑이 더욱 성능을 높이는 것으로 나타났다.

그리고 시뮬레이션의 검증을 위하여 LHT 블록 대체 정책에 대한 Markov 프로세스 모델을 제시, 분석하였으며 시뮬레이션에서와 일관된 결과를 보여주고 있다.

참 고 문 헌

1. Sung K. Chung, *Remote Procedure Call Design for Flexibility and Performance in Heterogeneous Environments*, PhD thesis, Univ. of Washington, 1990.
2. Niranjan G. Shivaratri and Phillip Krueger, "Two adaptive location policies for global scheduling algorithms," In *Proceedings of the 10th IEEE International Conference on Distributed Computing Systems*, pp. 502-509, 1990.
3. W. R. Stevens, *UNIX Network Programming*, Prentice-Hall, 1990
4. S. Yang and Y. Cho, "A Performance Study of a Connection Caching Technique," *Proceedings of the IEEE WESCANEX '95*, pp. 90-94, 1995.
5. Karedla R. et al, "Caching Strategies to Improve Disk System Performance," IEEE Computer, pp. 38-46, 1994.
6. ISO, ISO *8473 : Information Processing Systems-Addendum 3*
7. Per Jomer, "Connection caching of traffic adaptive dynamic virtual circuits," In *Proceedings of ACM SIGCOMM '89*, pp. 13-24, 1989.
8. S. Yang and Y. Cho, "Distributed Programming in More efficient Remote Shell Environment with Direct Data Forwarding and Lazy Connection Release," *Proceedings of the 14th IEEE International Phoenix Conference on Computers and Communications*, pp. 720-726, 1995.
9. W. R. Stevens, *TCP/IP Illustrated, Volume 1*. Addison-Wesley, 1994
10. J. Mogul, "Network Locality at the Scale of

Processes," In *Proceedings of ACM SIGCOMM '91*, pp. 273-284, 1991.

11. S. J.R.Caceres, P. B. Danzig and D. J. Mitzel, "Characteristics of Wide-Area TCP/IP Conversations," In *Proceedings of ACM SIGCOMM '91*, pp. 101-112, 1991.
12. K. S. Trivedi, Probability and Statistics with Reliability, Queuing, and Computer Science Applications, Prentice Hall, 1982
13. J. R. Spirn, Program Behavior: Models and Measurements, Elsevier North-Holland, 1977
14. S. Ross, Introduction to Probability Models, Academic Press, 1980
15. W. Reinhardt, "Advance Reservation of Network Resources for Multimedia Applications," Lecture Notes in Computer Science, Vol. 868, pp 23, 1994.
16. R. Bettati et al, "Connection Establishment for Multi-Party Real-Time Communication," In *Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Video*, 1995.
17. D. Ferrari, "A New Admission Control Method for Real-Time Communication in an Internetwork," S. Son, Ed., Advances in Real-Time Systems, Prentice Hall, 1995.

양 수 미(Soomi Yang) 정회원

1985년 : 서울대학교 컴퓨터공학과 학사

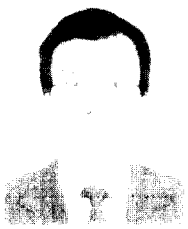
1987년 : 서울대학교 컴퓨터공학과 석사

1987년~1988년 : 시스템공학센터 (SERI) 연구원

1988년~현재 : 한국통신 연구개발원 전임연구원

1991년~현재 : 서울대학교 컴퓨터 공학과 박사과정

※주관심분야 : 운영체제, 분산 시스템



조 유 근(Yookun Cho) 종신회원

1971년 : 서울대학교 공과대학 건축학과 졸업

1978년 : 미국 미네소타대학 전산학과 박사학위 취득

1979년~현재 : 서울대학교 컴퓨터 공학과 교수

1984년~1985년 : 미국 미네소타대학 교환교수

1993년~1995년 : 서울대학교 중앙교육연구전산원장

1995년 : 한국정보과학회 부회장

※주관심분야 : 운영체제, 알고리즘의 설계와 분석