누적적 잉여용량 공유를 통한 이질적 다중 서버 시스템의 공정 스케줄링

박 경 호[†] · 황 호 영[††]

요 약

이 논문에서는 컴퓨터 시스템을 이질적 서버들로 구성된 시스템으로 간주하고, 장기적 관점에서 응용들간의 공정성을 추구하는 누적적(累積的) 공정 스케줄링 방법을 제시한다. 기존의 단일 서버 환경에서 주로 사용되는 GPS(generalized processor sharing) 기반의 스케줄링 알고리즘들은 순간적 관점에서 서버의 용량을 분배한다. 그러나 이를 이질적 다중 서버 환경에 적용하는 경우, 스케줄링 순서에 의한 지연시간의 오차가 서버들을 거치면서 누적될 수 있고, 잉여용량이 순간적 관점에서만 배분되기 때문에 장기적 관점에서 불공정성 문제가 발생할 수 있다. 본 논문의 방법에서는 각 응용의 예약용량을 보장하면서 잉여용량의 적절한 배분을 통해 장기적 관점의 공정 서비스를 추구한다. 이를 위해, 각 응용이 이상적으로 진행되기 위해 받아야 할 공정한 서비스 용량을 주기적 관찰을 통해 동적으로 파악하여 참조용량 모델로 삼고, 스케줄러는 응용들이 이 참조용량 모델을 점진적으로 따르도록 한다. 또한 이 모델을 효율적으로 구현하기 위한 휴리스틱 알고리즘을 만들고 실험을 통해 이를 검토한다.

키워드 : 공정 스케줄링, 이질적 다중 서버 시스템, 장기적 공정성, 누적적 잉여용량 분배

A Fair Scheduling of Heterogeneous Multi-Server Systems by Cumulative Extra Capacity Sharing

Kyeongho Park[†] · Hoyoung Hwang[††]

ABSTRACT

In this paper, we regard computer systems as heterogeneous multi-server systems and propose a cumulative fair scheduling scheme that pursues long-term fairness. GPS(generalized processor sharing)-based scheduling algorithms, which are usually employed in single-server systems, distribute available capacity in an instantaneous manner. However, applying them to heterogeneous multi-server systems may cause unfairness, since they may not prevent the accumulation of scheduling delays and the extra capacities are distributed in an instantaneous manner. In our scheme, long-term fairness is pursued by proper distribution of extra capacities while guaranteeing reserved capacities. A reference capacity model to determine the ideal progresses of applications is derived from long-term observations, and the scheduler makes the applications gradually follow the ideal progresses while guaranteeing their reserved capacities. A heuristic scheduling algorithm is proposed and the scheme is examined by simulation.

Key Words : Fair scheduling, Heterogeneous multi-server system, Long-term fairness, Cumulative extra capacity sharing

1. 서 론

공정한 서비스는 스케줄러의 주요한 요구사항 중의 하나이다. 이 논문에서는 단일 서버를 가정하고 만들어진 기존의 공정 스케줄링 알고리즘을 다중 서버 환경에 그대로 적용할 때 발생할 수 있는 문제점을 지적하고, 이질적 다중 서버 환경에서 장기적 관점의 공정 서비스를 추구하는 누적적 공정 스케줄링 방법을 제안한다.

일반적으로 하나의 컴퓨터 시스템은 다양한 이질적 서버들로 구성된 시스템으로 모델링될 수 있다. 예를 들면 네트워크 상의 파일 서버는 CPU 작업을 담당하는 CPU 서버, 파일 입출력을 담당하는 디스크 서버, 통신을 담당하는 네트워크 서버로 구성된 이질적 다중 서버 시스템이다. 그리고 응용은 서버들을 오가며 요청들을 발생시키므로, 요청들의 연속된 흐름으로 추상화할 수 있다.

각 응용이 가중치(weight)에 기반한 서비스 예약을 하고 예약한 서비스를 제공받는다면 해당하는 용량의 전용 서버에서 실행되는 것과 같은 효과를 누릴 수 있을 것이다. 한

※ 이 연구는 2007년도 한성대학교 교내연구비의 지원을 받아 수행되었습니다.
† 준 회 원 : 서울대학교 전기컴퓨터공학부 박사과정
†† 종신회원 : 한성대학교 멀티미디어공학과 조교수
논문접수 : 2006년 8월 10일, 심사완료 : 2007년 10월 15일

편 서버 입장에서는 이 가중치 이상의 서비스율을 제공함으로써 예약된 서비스를 보장할 수 있다. 일반적인 작업보전형(work-conserving) 서버는 수행할 작업이 있는 한 휴면상태로 들어가지 않고 계속 작업을 수행하므로 가중치에 따라 보장해야 하는 예약용량(reserved capacity) 뿐만 아니라 예약용량을 제공하고도 남는 용량인 잉여용량(extra capacity)도 서비스에 사용하는데, 이 잉여용량의 분배방법이 서비스의 전체적인 공정성을 꾀하는 데에 중요한 역할을 한다.

단일 서버에서의 공정 스케줄링 기법에 대해서는 네트워킹과 프로세스 스케줄링 분야에서 많은 연구가 있어 왔다. 기존의 대부분의 공정 스케줄링 알고리즘들의 기반이 되는 GPS(Generalized Processor Sharing)[1][2]에서는, 현재 큐에 적체된(backlogged) 응용들에게만 미리 예약된 가중치의 비에 따라 서버의 가용용량을 분배하는데 이 때 응용들은 원래 받아야 하는 예약용량 이외에 잉여용량도 가중치에 비례하여 추가로 제공받게 된다. 그리고 적체되지 않아 서비스를 받지 않은 응용은 추후에 그에 대한 별도의 보상을 받지 않는다. 이런 방식은 순간적 잉여용량 공유(instantaneous extra capacity sharing) 방식이라고 표현할 수 있다. 이 방법은 순간적 관점에서 보면 가장 공정한 서비스 방법이다. 그런데 여러 응용을 동시에 서비스할 수 있는 것으로 가정하는 이상적 모델인 GPS를 실제로 적용하기 위해서는, 한 시점에 한 응용에 대해서만 서비스를 하는 패킷형 모델용의 알고리즘들[1][2][3][4][5]을 사용하게 되므로, 스케줄링 순서에 따라서 각 응용은 제한된 범위내의 지연시간을 겪을 수 있다.

그런데 이질적 다중 서버 시스템에서 GPS에 기반한 알고리즘들을 각 서버에 개별적으로 적용할 경우, 전체적으로 보아 단일 서버 환경에서와 같은 공정한 서비스를 제공하지 못할 수 있다. 그 이유는 앞에서 언급했듯이 응용이 스케줄링 순서에 따른 지연시간을 겪는데, 여러 서버들을 거치며 수행되는 과정에서 이 지연시간으로 인한 오차가 누적될 수 있기 때문이다. 또한 GPS에서는 서버의 잉여 용량이 순간적인 관점에서 현재 서버에 적체된 응용들 사이에서만 분배되는데, 이 때 응용들이 제공받는 잉여용량은 그 시점의 큐의 상태에 종속적이다. 이러한 특성으로 인해, 특정한 응용이 장기적 관점에서 다른 응용들에 비해 상대적으로 많은 서비스를 받는 경우가 생길 수 있다.

이 논문에서는 이러한 문제점을 지적하고, 이를 장기적 관점에서 해결하기 위한 대안으로 다중 서버용의 새로운 공정 서비스 모델을 제안한다. 이 모델은 예약용량을 보장하면서 장기적 관점에서 누적적으로 잉여용량을 분배함으로써, 스케줄링 과정에서 발생하는 오차를 보완하고 잉여용량 측면에서도 공정한 스케줄링을 수행하기 위한 자원 관리 방법이다. 이 모델에서는 주기적인 관찰을 통해 각 응용에게 공정하게 주어져야 할 가상적인 용량 모델을 결정하는데, 이 모델은 응용이 이상적으로 진행될 수 있는 가상적인 전용의 서버를 의미하게 된다. 그리고 시스템은 예약용량을 보장하는 범위 내에서 응용들이 이 모델을 따를 수 있도록 스케줄하게 된다. 이 논문에서는 또한 이 모델의 구현을 위한 휴리스틱 알고리즘을 고안하고, 시뮬레이션을 통해 이를 검토한다. 요약하면, 제안하는 누적적 잉여용량 공유(cumulative extra capacity sharing) 방법은 단일한 서버만을 고려하는 기존의 방법에 비해 시스템의 전체적 관점에서 응용들이 어떻게 스케줄되어야 공정한지에 대한 관점을 제공해 주는 모델이다.

이후의 논문의 구성은 다음과 같다. 2장에서는 기존의 순간적 잉여용량 공유방법을 이질적 다중 서버 시스템에 적용할 때의 문제점을 살펴보고, 기존의 관련연구를 소개한다. 3장에서는 이 논문에서 제안하는 참조용량 모델의 의미와 그 역할에 대해서 설명한다. 4장에서는 구현을 위한 휴리스틱 스케줄링 알고리즘에 대해서 기술하며, 5장에서는 실험을 통한 결과를 보인다. 끝으로 6장에서는 요약 및 결론을 맺는다.

2. 순간적 잉여용량 공유의 문제점 및 관련연구

기존의 GPS 기법은 단일 서버 환경에서 모든 시점에서 순간적으로 공정한 서비스를 제공하기 위한 이상적 유체 모델(fluid model)이다. GPS에서 각 응용들은 보장된 서비스를 받을 수 있으며 타 서비스들과 무관하게 독립적인 서비스를 받는다. GPS에서는 현재 큐에 적체된 응용들이 미리 예약한 가중치, 즉 예약용량의 비에 따라 매 시점에 가용한 서버의 용량을 분배한다. 따라서 기본적으로 예약용량들이 보장되며, 이를 충족시키고 남은 잉여용량도 가중치의 비에 따라 분배된다. 가중치가 r_i인 응용 A_i가 구간 (t_1, t_2)에서 받은 서비스의 양을 $S_i(t_1, t_2)$라고 하면, GPS에서는 응용 A_i가 (t_1, t_2) 구간에서 연속 적체되어 있을 때

$$\frac{S_i(t_1,t_2)}{S_j(t_1,t_2)} \geq \frac{r_i}{r_j}, \quad j=1,2,...,N \tag{1}$$

을 만족한다[1].

그런데 GPS는 가상적인 유체 모델을 기반으로 가용용량을 무한히 쪼갤 수 있다는 가정하에서만 성립되는 이상적 방법이므로, 현실의 스케줄러에 그대로 적용할 수는 없다. 따라서, 이를 실제 서비스에 사용할 수 있도록 패킷 모델에 적용한 알고리즘들이 다수 개발되어 있다. PGPS(packetized generalized processor sharing)[1] 또는WFQ(weighted fair queueing)[2]는 이상적인 GPS 모델을 패킷 모델에 적용하기 위한 시도이다. PGPS는 유체 모델에서 각 요청의 시작 태그(start tag)와 종료 태그(finish tag)를 계산하여, 종료 태그의 값이 작은 것을 우선적으로 스케줄링한다. 응용 A_i의 k번째 패킷의 도착시간, 길이, 시작 태그, 종료 태그를 각각 a_i^k, L_i^k, S_i^k, F_i^k라고 가정하고, 시스템의 가상시간을 $V()$로 나타낼 때, 태그의 계산 방법은 다음과 같다.

$$S_i^k = \max\{F_i^{k-1}, V(a_i^k)\}$$

$$F_i^k = S_i^k + \frac{L_i^k}{r_i}$$

$$F_i^0 = 0$$

이후에 개발된 SCFQ(self-clocked fair queueing)[3], SFQ(start-time fair queueing)[4], WF^{2}Q(worst-case fair

weighted fair queueing)[5] 등의 다양한 방법들도 기본적으로 모두 GPS의 철학을 따르고 있으며, 이전 방법들의 구현 복잡도를 줄이고 지연시간 한계(delay bound)를 감소시키려는 시도를 한 것이다[6][7]. 유사한 연구인CBFS(credit-based fair scheduling)[8]는 일정시간마다 쌓인 크레딧을 서비스시에 사용하며 이의 많고 적음을 기준으로 스케줄링을 한다.

CPU 자원의 공정 분배에 사용되는 stride 스케줄링[9][10]은 PGPS의 방법론을 프로세스 스케줄링에 적용한 예이다. Hellerstein의 연구[11]는 UNIX에서 널리 사용되는 decay usage 스케줄러의 매개변수를 제어하여 공정한 자원 분배를 시도한 예인데, 모든 응용이 항상 CPU에 대한 요청을 내고 있어야 한다는 제약조건이 있다. 또한 디스크의 공정한 스케줄링을 위해서는 [12] 등의 연구가 있었다. 이러한 스케줄러들은 하나의 단일한 서버만을 대상으로 삼고 있으며 시스템 전체의 공정성을 고려하지는 않고 있다.

앞서 말했듯이 GPS를 실제로 구현한 알고리즘들은 이상적인 GPS와 달리 여러 응용을 동시에 처리할 수는 없으므로 서비스가 요청 단위로 순서대로 처리되는데, 실제 스케줄링된 순서에 따라 응용의 이상적인 종료시점과 실제 종료시점 간의 차이가 발생한다. 단일 서버 환경에서는 일시적인 불공정성이 다음 요청을 처리하는 과정에서 보상될 수 있기 때문에 이 오차는 일정 범위 이내로 제한되지만, 이질적 다중 서버 시스템에서와 같이 응용이 여러 서버를 오가며 수행되는 경우에는 이러한 오차들이 점차로 누적되어 결과적으로 특정 응용에 유리한 서비스 분배가 이루어질 수 있다.

또한 순간적 잉여용량 공유 방법에 기반한 GPS 방법에서는 다양한 도착 특성을 보이는 여러 응용이 혼재해 있는 경우 특정한 특성의 응용이 잉여용량을 상대적으로 많이 향유하는 상황이 생길 수 있다. 예를 들어 예약용량이 0.1씩인 두 응용 A_1, A_2를 가정해 보자. 응용 A_1은 서버 S_1에 대해서만 요청을 내는 데 반해, 응용 A_2는 서버 S_1과 S_2 사이를 번갈아 가며 요청을 낸다. 응용 A_1의 경우 서버 S_1을 독점하고 있는 동안은 모든 잉여용량을 사용하며, 응용 A_2와 함께 적체되어 있는 동안은 A_2와 같은 비율로 잉여용량을 받게 된다. 그러나 A_2는 어떤 시점에 도착하더라도 S_1의 용량의 절반만을 사용한다. 두 응용은 S_1에 대해 동일한 가중치를 지니고 있음에도 불구하고 장기적 관점에서 실제 받는 서비스의 비율은 다르다. 이 예에서 알 수 있듯이 같은 가중치를 가지는 응용도 서버의 적체 상태에 따라 받는 용량이 달라지고, 스케줄링 순서에 따라서 요청에 대한 종료시점도 달라진다. 따라서 여러 서버를 오가는 응용의 경우 다음 서버로 이동하는 시점이 이전 서버의 적체상태에 따라서 달라질 수 있다. 이는 요청이 서버에 도착하는 시점이 응용 자체에 의해서만 결정되던 단일 서버 환경과는 다른 상황이다.

이 논문에서 제안하는 모델에서는 순간적 시점의 서버의 적체상태에 의해서만 용량 분배가 이루어지는 방법에서 벗어나, 각 응용이 각 서버들에서 받는 서비스의 양을 관찰하여 평균적으로 받아야 하는 참조용량을 결정한다. 그리고 다른 응용의 간섭이 없이 각 응용에게 참조용량만큼의 전용 서버가 주어지는 것과 같은 이상적인 상황을 가정하고 이 때의 서비스 진행을 모델링하여 이를 참조한 스케줄링을 수행한다.

다중 서버 시스템의 공정 스케줄링에 대해서도 기존에 연구가 있어 왔는데, 이는 다시 동질적 다중 서버 시스템(homogeneous multi-server system)과 이질적 다중 서버 시스템(heterogeneous multi-server system)으로 구분할 수 있다. 동질적 다중 서버 시스템에서의 공정 스케줄링에 관한 연구로는 SFS(Surplus Fair Scheduling)[13]를 예로 들 수 있다. SFS에서는 다중프로세서 시스템에서 쓰레드를 스케줄링할 때 실제 요구된 가중치들을 가능한(feasible) 가중치로 재조정하고 각 프로세서를 독립적인 GPS 서버로 모델링하여 스케줄링한다. 그러나 이 방법은 모든 서버가 동질적이므로 특정 요청이 특정 서버에서 수행되어야 한다는 제약이 없기 때문에, 본 논문에서 대상으로 하는 이질적 다중 서버 모델에는 적합하지 않다.

이질적 다중 서버 모델을 대상으로 하는 연구로는 MTR-LS(Move-To-Rear List Scheduling)[14], PCFQ (Prediction-based Composite Fair Queueing)[15]와 같은 예가 있다. MTR-LS의 경우, 스케줄링 지연시간의 합이 상수로 한정되도록 함으로써 지연시간의 오차가 누적되는 문제를 해결하고 있으나, 이 방법에서는 공정성은 예약용량의 관점에 한정되어 있으며 잉여용량에 대해서는 별도로 고려하지 않으므로 특정 응용이 잉여용량을 많이 받음으로써 여전히 불공정한 상황이 발생할 수 있다. PCFQ의 경우, 모든 요청이 미리 정해진 서버를 고정된 순서대로 거쳐 처리된다고 가정하고 초기에 한번만 스케줄링하고 각 서버에서는 FIFO (first-in first-out) 순서로 서비스하는 방법으로서 주로 네트워크 라우터처럼 각 패킷이 CPU를 거쳐 바로 네트워크로 보내지는 식의 특정한 응용에 적합하며, 모든 서버에서 개별적으로 스케줄링이 이루어지는 본 연구와는 가정상의 근본적인 차이가 있다.

3. 참조용량 모델

이 절에서는 이 논문에서 제시하는 스케줄링 기법의 바탕이 되는 참조용량 모델의 의미와 이를 구하는 방법에 대해 설명한다.

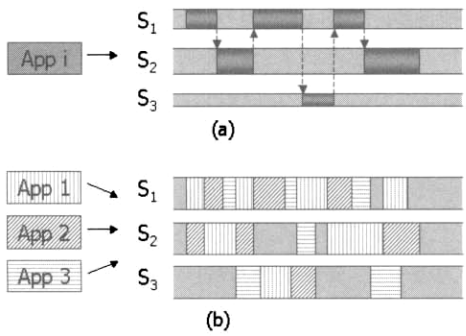
이하의 논의에서는 다음과 같은 가정을 한다. 시스템은 다수의 이질적인 서버들 S_j로 구성되어 있다. 각 서버들의 용량은 편의상 1로 가정한다. 각 응용 A_i는 서버 S_j에 대해 예약용량 r_i^j를 가진다. 응용 A_i는 서버들에게 일련의 연속된 작업요청을 내며, 이는 응용이 서버들 사이를 오가는 것으로 해석할 수 있다. 그리고 서버들은 적체되어 있는 응용이 있는 한 계속 서비스를 하는 작업보전형으로 동작한다. 따라서 한 서버에 대해서 현재 적체된 응용들의 예약용량들의 총합이 1보다 적으면 잉여용량이 생기게 되며 응용들은 이것을 활용할 수 있다.

개별적인 응용의 관점에서 보면, 예측가능한 서비스를 받

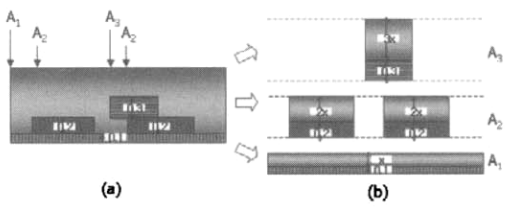
기 위해서는 (그림 1(a))에서처럼 모든 서버들로부터 전용의 고정적인 용량을 제공받는 것이 이상적일 것이다. 한 응용에 주어지는 서버의 이상적 용량이 알려져 있다면, 요청의 도착시간과 전용용량을 통해 각 요청의 종료시간을 구할 수 있고, 이러한 응용의 진행이 이상적이며 공정한 모델이라고 할 수 있다. 그러나 각 서버에서의 응용들의 적체 상황에 따라서 응용이 받는 용량과 서비스 순서가 바뀌기 때문에 현실의 시스템에서는 이러한 이상적인 진행이 그대로 재현될 수는 없다. 따라서 본 방법에서의 스케줄러의 역할은 일단 이상적인 진행 모델을 만들고, 실제의 스케줄과 이상적 스케줄 사이의 차이를 최소화하면서 현실의 응용이 이상적인 진행을 최대한 따를 수 있도록 스케줄링하는 것이다.(그림 1(b)) 참조용량(reference capacity)이란 요청의 도착시간과 무관하게 응용이 기대하는 가상적인 전용 용량을 의미한다. 참조용량은 미리 주어진 예약용량과 실행 과정에서 부가적으로 얻게 되는 잉여용량의 합으로 구성된다. 응용이 받는 이상적인 잉여용량은 한 적체 구간 안에서 일정해야 하며, 공정성을 위해서는 임의의 두 응용에 대해서 그들의 예약용량에 비례해야 타당할 것이다.

따라서 응용 A_i가 서버 S_j로부터 받는 잉여용량 $e_i^j = a_j r_i^j$로 표현할 수 있다. 여기에서 a_j는 S_j마다 따로 결정되는 값이다. 그러면 A_i가 S_j에서 받을 참조용량 c_i^j는 $r_i^j + e_i^j = (1+a_j)r_i^j$이다. 다시 말해, 각 응용이 받는 이상적인 참조용량들의 비는 그들의 예약용량의 비와 같아야 한다.

(그림 2)는 예약용량이 각각 0.1, 0.2, 0.3인 세 개의 응용 A_1, A_2, A_3의 예이다. (a)에서 잉여용량은 응용들 사이에서 순간적으로 공유된다. 하지만 제안하는 모델에서는 (b)에서처럼 각 응용이 다른 응용에 무관하게 각각 가상적인 전용의 참조용량을 가지는 것으로 간주한다. 이 때 그림의 x



(그림 1) (a) 응용의 관점에서 본 이상적 모델
(b) 스케줄러의 관점에서 본 응용들의 진행



(그림 2) (a) 순간적 잉여용량 공유 모델
(b) 각 응용의 참조용량 모델

는 응용 A_1에게 주어지는 이상적인 잉여용량을 의미한다.

결국 a_j의 값은 서버의 전체 용량에서 잉여용량이 차지하는 비중을 의미하며, 서버 S_j 내의 모든 응용들에 공통적으로 적용된다. 이 값은 응용들이 발생하는 요청의 양에 따라 동적으로 변화하므로 관찰을 통해서 도출한다. 이를 위해 장기간 동안 서버에서 발생하여 이용된 잉여용량을 측정하고 각 응용에 할당된 평균 잉여용량을 계산한다.

정리. 순간적인 잉여용량 공유 모델에서 a_j는 다음을 측정하여 구할 수 있다.

$$\frac{\sum \text{잉여용량으로 서비스된 작업}}{\sum \text{예약용량으로 서비스된 작업}}$$

증명. 순간적 잉여용량 공유 모델에서 서버 S_j에서 응용 A_i가 예약용량과 잉여용량으로 받은 서비스의 양을 각각 $R_{i,j}^I$, $E_{i,j}^I$라고 하자(그림 2(a) 참조). 그리고 참조용량 모델에서 응용 A_i가 예약용량과 잉여용량으로 받은 서비스의 양을 각각 $R_{i,j}^R$, $E_{i,j}^R$이라고 하자(그림 2(b) 참조). 그러면 임의의 I에 대해

$$a_j = \frac{e_i^j}{r_i^j} = \frac{E_{i,j}^R}{R_{i,j}^R} = \frac{\sum_i E_{i,j}^R}{\sum_i R_{i,j}^R} \tag{2}$$

가 성립하고, 정의에 의해

$$\sum_i E_{i,j}^R = \sum_i E_{i,j}^I \text{ and } \sum_i R_{i,j}^R = \sum_i R_{i,j}^I \tag{3}$$

이다. 따라서

$$a_j = \frac{\sum_i E_{i,j}^I}{\sum_i R_{i,j}^I} \tag{4}$$

□

위의 값의 정확한 계산을 위해서는 유체 서버의 에뮬레이션이 요구된다. 그러나 실제 스케줄링시에 이 계산을 하는 것은 과도한 부담이 될 수 있으므로 구현상의 편의를 위해 유체 서버를 근사하는 패킷 서버 모델이 사용될 수 있다. 즉 유체 모델에서의 적체 구간을 계산하는 부담을 피하기 위해 실제 스케줄링 과정에서 요청이 적체되어 있는 구간을 측정하고 $R_{i,j}^I$ 의 계산에 이 값을 대신 사용한다. 그러면

$$\sum_i E_{i,j}^I + \sum_i R_{i,j}^I = \sum_{all\_req\_in\_Sj} L \tag{5}$$

이므로

$$a_j = \frac{\sum_{all\_req\_in\_Sj} L}{\sum_{all\_req\_in\_Sj} r_i^j (t^{finish} - t^{arr})} - 1 \tag{6}$$

가 된다. 이 때 L은 요청의 길이이고, t^{arr}과 t^{finish}는 각각 요청의 도착시간과 종료시간이다. 이 경우 $R_{i,j}^I$가 실제 값보다 적게 측정되므로, a_j는 정확한 값보다 약간 크게 계산된다.

a_j는 주기적인 측정과정에서 발생하는 오차를 보정해야 하고, 서버의 상태가 바뀔 경우 이를 반영해야 하므로 주기적으로 갱신된다. 이 값의 급격한 변화를 막기 위해 지수적

평균화(exponential averaging)를 통해 점진적으로 갱신한다.

$$a_j^{new} = ca_j^{old} + (1-c)a_j^{measured}, 0 \le c \le 1 \quad (7)$$

이 때 a_j^{new}, a_j^{old}, $a_j^{measured}$는 각각 새로 계산되는 값, 이전의 값, 관찰을 통해 얻어진 값이고, c는 a_j 갱신의 점진성을 결정하는 상수이다.

4. 구현을 위한 알고리즘

3절에서 설명한 바와 같은 참조용량 서버가 주어지면, 시간 t^{arr}에 도착하는 요청의 이상적인 종료시간, t^{ic}를 다음과 같이 계산할 수 있다.

$$t^{ic} = t^{arr} + \frac{L}{(1+a_j)r_i^j} \quad (8)$$

그러나 실제 스케줄에서는 패킷 형태의 스케줄링으로 인해 발생하는 지연시간 때문에 응용의 진행과 이상적인 참조 모델 사이에 차이가 발생한다. t^{ic}와 실제 종료시간 t^{rc}의 차이를 δ로 표현하면

$$\delta = t^{ic} - t^{rc} \quad (9)$$

이다. δ는 응용이 이상적인 진행 모델에 근거해 기대했던 것보다 빠르거나 느린 정도를 나타낸다. 공정한 스케줄링을 위해서는 응용의 진행이 이상적인 진행 모델을 따를 수 있도록 해 주어야 한다. 지난 번 요청의 실행 결과로 δ가 발생했다면, 이상적인 진행을 위해서는 이상적인 종료 시간을 다음과 같이 보정해 주어야 한다.

$$t^{ic} = t^{arr} + \frac{L}{(1+a_j)r_i^j} + \delta \quad (10)$$

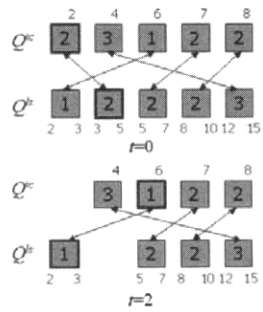
스케줄러는 t^{ic}가 가장 적은 요청이 가장 긴급한 것이므로 이를 선택하여 서비스하면 된다. 그러나 스케줄러는 또한 δ 값이 큰 응용이 서버를 독점하여 다른 응용에게 기아현상(starvation)을 일으키는 것을 막아야 한다. 이는 기본적으로 각 응용의 예약용량을 지켜주면 자연스럽게 해결되는 문제이다. 예약용량을 보장하기 위해 요청이 시작되어야 하는 가장 늦은 시간을 t^{ls}라고 하면

$$t^{ls} = t^{arr} + \frac{L}{r_i^j} - L \quad (11)$$

이다. 따라서 기본적인 스케줄링 원칙은 다른 응용의 t^{ls}를 침해하지 않는 범위 내에서 가장 적은 t^{ic} 값을 가지는 응용을 선택하는 것이다.

효율적인 구현을 위해 본 논문에서는 위와 같은 원칙에 기반을 둔 휴리스틱 알고리즘을 사용한다. 이 알고리즘에서는 두 개의 큐, Q^{ic}와 Q^{ls}가 유지되는데 서버에 도착한 모든 요청은 양쪽 큐에 각각 정해진 규칙에 의해 동시에 저장된다. 큐 Q^{ic}에는 요청들이 t^{ic}의 오름차순으로 저장된다. 또 다른 큐 Q^{ls}에는 요청들이 t^{ls}의 오름차순으로 저장된다. 스케줄러는 Q^{ls}의 머리에 있는 요청의 마감시간을 침해하지 않는다는 조건 하에 Q^{ic}의 머리에 있는 요청을 선택한다. 선택된 요청은 양쪽 큐에서 제거된다.

(그림 3)에 스케줄링의 한 예를 보였다. 각 상자 안의 숫자는 요청의 길이이다. t=0일 때 Q^{ic}의 첫번째 요청은 Q^{ls}의 첫 번째 요청의 t^{ls}를 침해하지 않으므로 스케줄될 수 있다. 그리고 나서 이 요청은 양쪽 큐에서 모두 제거된다. 그러나 t=2일 때, Q^{ic}의 첫번째 요청을 실행하게 되면 Q^{ls}의 첫번째 요청을 침해하게 되므로, 이 경우는 Q^{ls}의 첫번째 요청을 스케줄링한다.



(그림 3) 스케줄링의 예

5. 실 험

이 절에서는 3절의 모델을 4 절에서 만든 휴리스틱 알고리즘을 사용하여 시뮬레이션을 통해 검토한다. 각 서버의 a_j 값을 계산하기 위해서는 근사 패킷 모델을 사용한다. a_j의 갱신 주기는 1000 단위시간이며, c는 0.5를 사용한다.

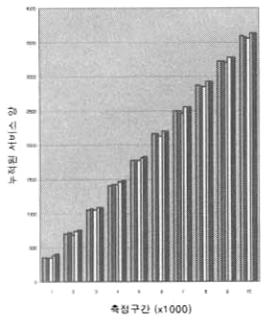
실험에는 두 종류의 요청 패턴이 사용된다. 한 가지는 요청의 길이와 이 요청들의 서버간 이동 확률이 균등 분포를 가지는 무작위 요청 패턴이고, 다른 한 가지는 요청의 길이와 이 요청들의 서버간 이동 순서가 고정되어 있는 반복 요청 패턴이다.

(그림 4)는 다섯 개의 응용 A_1, A_2, A_3, A_4, A_5에 제공되는 서비스의 누적된 양을 보인 것이다. 각 응용의 예약용량은 모두 0.1씩으로 가정한다. 요청은 무작위 패턴이 사용되었으며, 한 요청의 최대 길이는 10으로 가정한다. 이 결과에서 각 응용에 제공되는 서비스의 양이 거의 비슷한 정도를 유지하면서 증가함을 볼 수 있다.

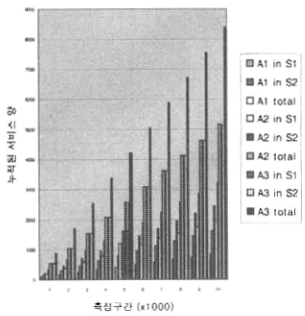
(그림 5)는 요청 패턴이 주기적인 경우에 누적되는 서비스의 양을 보인 것이다. 개별적인 부하를 보이기 위해 응용이 각각의 서버에서 받는 서비스의 양도 함께 보였다. 응용 A_1, A_2, A_3의 예약용량은 각각 0.1, 0.2, 0.3로 가정한다. 두 개의 서버 S_1, S_2에 대한 A_1의 부하 비율은 1:2 로 가정하며, A_2의 경우 3:5, A_3의 경우 5:8로 가정하는데, 이 값들은 임의로 선택된 것들이다. 이 결과를 통해 각 응용에게 주어지는 총 서비스의 합은 이 응용들의 예약용량에 거의 비례함을 볼 수 있다.

(그림 6)은 위와 동일한 설정에서 서버 S_1에서 응용들이

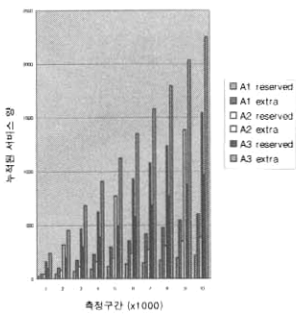
예약용량과 잉여용량으로 서비스된 양을 나타낸 것이다. 이 그림의 결과는 예약된 값들에 반드시 비례하지는 않는 것을 볼 수 있는데, 이는 서버에 주어지는 응용의 부하의 차이에 기인한다. 그러나 서비스 양의 분포는 거의 일정하게 유지되는 것을 관찰할 수 있다.



(그림 4) 실험 1



(그림 5) 실험 2



(그림 6) 실험 3

6. 결 론

이 논문에서는 이질적 다중 서버 시스템에서의 누적적 공정 스케줄링 기법을 제안하였다. GPS처럼 순간적 관점에서 용량을 공유하는 방법에 기반한 기존의 알고리즘들을 이질적 다중 서버에 그대로 적용하는 것이 적합하지 않다는 것을 지적하고, 이를 해결하기 위해 장기적 관점에서의 공정성을 추구하는 누적적 용량 공유 방법을 제시하였다. 이 방법에서는 이상적인 참조용량 서버 모델을 만들고 이를 통해 응용의 이상적인 진행과정을 도출하고 실제의 응용이 이를 따를 수 있도록 스케줄링하는 방법을 사용한다. 그리고 이를 구현하기 위한 휴리스틱 알고리즘을 만들고, 시뮬레이션을 통해 검토하였다.

참 고 문 헌

[1] A. K. Parekh and R. G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case." *IEEE/ACM Trans. on Networking*, Vol.1, No.3, pp.344-357, Jun. 1993.

[2] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," *ACM SIGCOMM '89*, pp.1-12, 1989.

[3] S. Golestani, "A Self-clocked Fair Queueing Scheme for Broadband Applications," *IEEE INFOCOM '94*, pp.636-646, 1994.

[4] P. Goyal, H. M. Vin, and H. Cheng, "Start-Time Fair Queueing: A Scheduling Algorithm for Integrated Services Packet Switching Networks," *IEEE/ACM Trans. on Networking*, Vol.5, No.5, pp.690-704, Oct. 1997.

[5] J. C. R. Bennett and H. Zhang, "Hierarchical Packet Fair Queueing Algorithms," *ACM SIGCOMM '96*, pp.143-156, 1996.

[6] H. Zhang, "Service Disciplines for Guaranteed Performance Service in Packet-Switching Networks," *Proc. of IEEE*, Vol.83, No.10, pp.1374-1396, Oct. 1995.

[7] P. Goyal and H. M. Vin, "Generalized Guaranteed Rate Scheduling Algorithms: A Framework, "*IEEE/ACM Trans. on Networking*, Vol.5, No.4, pp. 561-571, Aug. 1997.

[8] D. Pan and Y. Yang, "Credit Based Fair Scheduling for Packet Switched Networks," *IEEE INFOCOM 2005*, pp. 843-854, 2005.

[9] C. A. Waldspurger and W. E. Weihl, "Stride Scheduling: Deterministic Proportional-Share Resource Management," *Tech. Rep. MIT/LCS/TM-528*, MIT, 1995.

[10] D. L. Moal, M. Masuda, M. Goshima, S. Mori, Y. Nakashima, T. Kitamura, and S. Tomita, "Priority Enhanced Stride Scheduling," *IPSJ Trans. on HPCS*, Vol.43, No. SIG 6(HPS 5), pp. 99-111, Sep. 2002.

[11] J. L. Hellerstein, "Achieving Service Rate Objectives with Decay Usage Scheduling," *IEEE Trans. on Software Engineering*, Vol.19, No.8, pp.813-825, Aug. 1993.

[12] S. R. Seelam, *Towards Dynamic Adaptation of I/O Scheduling in Commodity Operating Systems*, Ph.D. dissertation, Univ. of Texas at El Paso, 2006.

[13] A. Chandra, M. Adler, P. Goyal, and P. Shenoy, "Surplus Fair Queueing: A Proportional-Share CPU Scheduling Algorithm for Symmetric Multiprocessors," *USENIX OSDI 2000*, pp. 1-14, 2000.

[14] J. Bruno, E. Gabber, B. Özden, and A. Silberschatz, "Move-To-Rear List Scheduling: a new scheduling algorithm for providing QoS guarantees," *ACM Multimedia '97*, pp. 63-73, 1997.

[15] F. Sabrina, S. S. Kanhere, and S. K. Jha, "Design, Analysis, and Implementation of a Novel Multiple Resource Scheduler," *IEEE Trans. on Computers*, Vol. 56, No.8, pp. 1071-1086, Aug. 2007.

박 경 호

e-mail : kalynda@archi.snu.ac.kr
1991년 서울대학교 컴퓨터공학과 학사
1993년 서울대학교 컴퓨터공학과 석사
1993년~현재 서울대학교 전기컴퓨터 공학부 박사과정
관심분야 : 스케줄링, 멀티미디어시스템 등

황 호 영

e-mail : hyhwang@hansung.ac.kr
1993년 서울대학교 컴퓨터공학과 공학사
1995년 서울대학교 컴퓨터공학과 공학석사
2003년 서울대학교 전기컴퓨터공학 공학박사
2003년~2007년 안양대학교 디지털미디어학부 조교수
2007년~현재 한성대학교 멀티미디어공학과 조교수
관심분야 : 정보통신, 무선 및 이동통신망, 멀티미디어시스템 등