석사학위논문
지도교수 이 민 석

리눅스에서의 Percentile 스케줄 기법 구현

An Implementation of Percentile Scheduler on Linux

2002 년 8 월 일

한성대학교 컴퓨터 신기술 대학원
컴퓨터 신기술 학과
컴퓨터 공학 전공
최 동 준

석사학위논문
지도교수 이 민 석

리눅스에서의 Percentile 스케줄 기법 구현

An Implementation of Percentile Scheduler on Linux

2002 년 8 월 일

위 논문을 공학석사 학위논문으로 제출함

한성대학교 컴퓨터 신기술 대학원
컴퓨터 신기술 학과
컴퓨터 공학 전공
최 동 준

최동준의 공학석사 학위논문을 인정함

2002 년 8 월 일

심사위원장 (인)

심사위원 (인)

심사위원 (인)

목 차

그림 목차

요 약

대화형 멀티미디어와 가상 환경과 같은 어플리케이션이 실행되기 위해서는 운영 체제가 실시간 계산과 통신 서비스를 제공하여야 한다. 점차적으로 이들 어플리케이션은 점차 전문적인 실시간 방식보다는 일반적 용도의 운영 체제에서 운영되므로 실시간 운영 기술을 일반적인 서버나 데스크탑 시스템에 적용하는데 커다란 관심이 집중되고 있다.

이 논문에서는 패킷 스케쥴에 사용된 Percentile 스케쥴 기법을 웹 서버에 적용하여 서버에 들어오는 다양한 서비스 요구를 서로 다른 클래스로 분류하여 차등적인 서비스를 제공하고자 하는 연구를 기술하였다. 논문에서는 범용 운영 체제를 실시간화 하기 위한 노력들을 살펴보았으며, Percentile에 기반한 스케쥴러를 리눅스 환경에 구성한 방법을 기술하였다. 제안된 기법의 성능을 살펴보기 위하여 실험을 수행하였으며 Percentile 스케쥴 기법을 적용한 경우 지정된 클래스에 따라 확연히 다른 응답 시간을 관찰할 수 있었다.

1.서 론

대화형 멀티미디어와 가상 환경과 같은 어플리케이션이 실행되기 위해서는 운영 체제가 실시간 계산과 통신 서비스를 제공하여야 한다. 점차적으로 이들 어플리케이션은 점차 전문적인 실시간 방식보다는 일반적 용도의 운영 체제에서 운영되므로 실시간 운영 기술을 일반적인 서버나 데스크 탑 시스템에 적용하는데 커다란 관심이 집중되고 있다.

최근 리눅스 커널에 기초를 둔 실시간 프로젝트가 시작되었다. 리눅스를 사용하는 이유는 오픈 소스 정책을 사용하여 넓은 개발자, 관찰자, 연구자가 있기 때문에 상업적 운영체제 보다 빠르게 문제점을 찾을 수도 있고 문제점에 대해서 빠른 해결방법을 무료로 지원을 받을 수 있다. 이런 이유로 리눅스는 단시간 안에 가장 유명한 운영 체제 중 하나가 되었다. 비록 어떤 리눅스 사용자들은 마이크로 소프트의 제품을 회피하는 의미에서 사용하기도 하지만 많은 리눅스 사용자들은 리눅스가 허용하고 있는 오픈 소스 정책으로 인한 안정적이고 안전하며 성능이 좋은 것에 끌리고 있다. 하지만 리눅스를 비롯한 많은 운영 체제 제품들이 정교한 스케줄러와 유연성 있는 자원 관리를 제공하지 않고 특정한 타이밍을 맞추어 주어야 하는 실시간 태스크를 지원하도록 설계되지 않았다.

범용 운영 체제를 사용하는 많은 시스템에서도 서비스 품질의 보장에 대한 요구가 커지고 있다. 특히 웹 서비스의 형태로 제공되는 멀티미디어 서비스, 차별화된 홈페이지 참조 등은 서비스 운영자의 수익에 많은 영향을 끼치도록 되어있다.
현재 일반적 용도의 운영 체제에서 실시간 서비스를 제공하거나, 적어도 서비스 특성상 요구되는 응답 시간을 100% 가까운 확률로 지키는 데는 커다란 문제점이 있다. 그 이유는 비주기적이거나 대화형인 응용 프로세스들의 비예측성과 운영 체제 자체의 구조적 복잡함, 비선점형 스케쥴, 인터럽트 처리나 프로토콜 처리에 주어지는 높은 우선 순위 때문에 실시간 시스템이 일반적으로 필요로 하는 시간 제약성을 만족시키는 것이 거의 불가능하기 때문이다. 또한 멀티미디어 계산 환경에서는 경성 실시간 시스템이 요구하는 엄격한 시간 제약성과는 달리 통계적인 예측성의 보장에 의한 시간 제약성을 요구하고 있다. 최근의 멀티미디어 서버와 같은 시스템

은 클라이언트에 사용되는 버퍼 메모리 요량의 증가에 힘입어 어느 정도의 시간 제약성 완화가 가능하며 이런 긍정적인 변화는 서버에게 경성이 아닌 통계적 예측성 보장으로 거의 같은 품질의 서비스를 제공할 수 있도록 하고 있다.

이 연구에서는 패킷 스케쥴에 사용되는 Percentile 스케쥴 기법을 리눅스 웹 서버에 적용하여 서버에 들어오는 다양한 서비스 요구를 분류하여 유형 별로 차등적인 서비스를 제공하는데 이용하고자 한다. 이 연구는 서버 호스팅 업체, 전자 상거래 사이트 등에서 모든 고객이 평등하지 않다는 현실 인식에 그 바탕을 두고 있다. 서버 호스팅의 예를 들면, 서로 다른 비용을 지불한 호스팅 고객에 대하여 같은 수준의 서비스를 제공하는 것은 공정하지 못하다. 많은 비용을 지불한 고객의 홈페이지에 대한 참조에 비하여 적은 비용을 지불한 고객의 홈페이지 보다 더 짧은 응답을 제공하여 호스팅 업체는 같은 서버 안에서 차별화된 서비스를 할 수 있게 된다. 또 전자 상거래 사이트의 경우, 사이트를 단순히 살펴보기 위하여 방문한 단발성 고객과 자주 물건을 구매한 고객, 또 그들 중에서도 물건을 구매하기 위하여 쇼핑 카트에 상품을 넣은 고객 등에 대하여 다른 품질의 서비스를 제공함으로써 더 많은 구매를 유도할 수 있다. 이러한 서비스 차별화 정책은 리눅스와 같은 범용 운영 체제가 주로 사용되는 환경에 특별한 프로세스 스케쥴 기법을 필요로 한다.

Percentile 스케쥴 기법은 경로 배정기에서 패킷 전송에 따른 우선 순위를 적용하기 위하여 개발된 스케쥴 방법이다[1]. 이 논문에서는 엄격한 예측성 보장이 요구되는 실시간 시스템이 아닌 환경에서 통계적 실시간성을 보장할 수 있는 Percentile 보장 스케쥴 기법을 제안하고 그 방법을 리눅스 웹 서버에 적용하여 위와 같이 차별화된 웹 서비스를 제공하는 용도로 사용하고자 한다.

이 논문의 구성은 다음과 같다. 제 2 장에서는 범용 운영 체제를 실시간 시스템에 사용하기 위한 노력들과 차별화된 웹 서비스에 관한 관련 연구를 살펴보고, 제 3 장에서는 Percnetile 시스템에 관해서 기술하고, 제 4 장에서는 리눅스의 스케쥴러에 관한 개괄적 이해와 우리가 제안하는 Percentile 스케쥴러에 관하여 기술한다. 제 5 장에서는 제 4 장에서 제안

된 방법을 구현하여 그 성능을 확인하기 위한 실험과 결과를 제시하고 제 6 장에서 결론과 추후 연구를 기술한다.

2. 관련 연구

현재 일반적 용도의 운영 체제에서 실시간 서비스를 제공하거나, 적어도 서비스 특성 상 요구되는 응답 시간을 100% 가까운 확률로 지키는 데는 커다란 문제점이 있다. 그 이유는 비주기적 이거나 대화형인 응용 프로세스들의 비예측성과 운영 체제 자체의 구조적 복잡함, 비선점형 스케쥴, 인터럽트 처리나 프로토콜 처리에 주어지는 높은 우선 순위 때문에 실시간 시스템이 일반적으로 필요로 하는 시간 제약성을 만족시키는 것이 거의 불가능하기 때문이다. 또한 멀티미디어 계산 환경에서는 경성 실시간 시스템이 요구하는 엄격한 시간 제약성과는 달리 통계적인 예측성의 보장에 의한 시간 제약성을 요구하고 있다. 최근의 멀티미디어 서버와 같은 시스템은 클라이언트에 사용되는 버퍼 메모리 요량의 증가에 힘입어 어느 정도의 시간 제약성 완화가 가능하며 이런 긍정적인 변화는 서버에게 경성이 아닌 통계적 예측성 보장으로 거의 같은 품질의 서비스를 제공할 수 있도록 하고 있다.

2.1 실시간 시스템

실시간 시스템이라 함은 실시간 태스크들로 구성되는 시스템을 일컫는데 실시간 태스크들은 대기 시간, 마감 시간, 그리고 수행주기와 같은 시간 제약 조건들을 가진다. 따라서 태스크의 수행 결과는 논리적인 정확성 뿐만 아니라 주어진 시간 제약 조건들을 만족시켜야만 올바르다고 할 수 있게 된다. 그러나 모든 태스크들이 시간 제약 조건들을 만족시켜야만 하는 실시간 시스템의 개발은 이로 인해서 많은 비용을 필요로 하게 된다. 또한 실시간 시스템은 물리적인 공정(physical process)을 제어하고 감독하는 시스템으로, 외부 센서(external sensor)들로부터 정보들을 모으고, 이를 처리하여 어떤 행동을 취하는 시스템으로 실시간 시스템에서 시스템이 반응을 하려면 물리적인 프로세스가 엄격한 시간 제한을 지켜야 한다. 만약 이러한 엄격한 시간 제한이 지켜지지 않으면 감지기가 모은 모든 정보가 없어지거나 낡은 정보가 되어 버리고, 결국 취해지는 행동 자체도 더 이상 적당하지 않게 된다. 이러한 것이 실시간 시스템의 마감 시간(deadline)이라는 개념을 유도하며, 실시간 시스템과 시분할 시스템과의 차

이점이라고 수 있다. 따라서 실시간 시스템에서 태스크의 마감 시간이란, 태스크의 실행이 꼭 종료되어야 하는 시간이다.

오늘날 하드웨어 가격의 급속한 하락으로 인하여 대부분의 기업에서 실시간 시스템과 장비를 광범위하게 적용할 수 있게 되었다. 그 적용 분야로는 공정제어, 공장의 자동화, 의학 및 과학 연구, 컴퓨터 그래픽, 지역 및 광역 통신, 우주 시스템, 전화국의 전화 교환기, 공항의 항로 관제 시스템, 은행의 안전 감시 시스템, 국방 지휘 통제 시스템, 가정 자동화 시스템 등의 공업, 상업 그리고 국방 부문에서 널리 쓰인다. 이 시스템들은 다수의 독립적인 입력 사건을 처리해야 하며, 다수의 출력을 생성해야 하므로 매우 복잡하다. 또한 이 사건들은 예측할 수 없는 간격으로 발생한다. 그럼에도 불구하고 이 시스템들은 소프트웨어 요구 사항에서 기술된 시간적 제약 내에 응답해야 한다. 그리고 입력 부하가 시간에 따라 심하고 불규칙하게 변동한다. 즉, 사용자의 생명에 관련되거나 장비 안에 내재되어 있거나(자동차의 자동연료 분사 시스템) 통신용이거나 센서를 통해 아날로그 신호를 받아들이는 시스템들은 대부분 실시간 시스템으로 간주할 수 있다. 실시간 시스템은 마감 시간 이후 결과의 유효성에 따라 크게 경성 실시간 시스템(hard real-time system)과 연성 실시간 시스템(soft real-time system)으로 나뉘어진다.

2.1.1 경성 실시간 시스템

경성 실시간 시스템에서 각 태스크는 경성 마감 시간을 가진다. 경성 마감 시간을 갖는 태스크란, 태스크의 실행이 마감 시간 이내에 종료되지 않는다면 그 태스크는 시간에 의한 결함을 발생시키며, 이러한 시간 결함은 전체 시스템의 붕괴를 야기 시킨다. 이런 예로 자동화된 공장, 발전소 제어, 원격 탐사 등이 있으며 이런 곳에는 처리 결과가 정확해야 할 뿐만 아니라 정해진 시간 내에 그 결과를 얻을 수 있어야 하고 만일 그렇지 않으면 시스템에게는 치명적이다.
한 가지 예로 원자력 발전소에서 과열된 원자로를 냉각시키는 태스크가 만약 마감 시간 후에 반응한다면 엄청난 결과를 발생시킨다. 즉, 경성 실시간 시스템은 효율성에 대한 고려는 별로 중요하지 않으며 시간 제약성의 만족이 중요하다. 그러므로 운영 체제에 각 태스크의 마감 시간 만족을 강요되며, 마감 시간을 지키지 못하는 경우 치명적인 결과를 초래할 수 있

다.

2.1.2 연성 실시간 시스템

연성 실시간 시스템의 태스크들은 연성 마감 시간을 갖는다. 태스크가 연성 종료 시한을 갖는다는 말은, 태스크가 마감 시간 내에 실행이 종료되지 못해도 약간의 시스템 성능 저하를 초래할 뿐 완전한 시스템의 붕괴로까지 이르지 않는 시스템을 연성 실시간 시스템이라 한다. 연성 실시간 시스템에서 태스크가 마감 시간 이후에 종료되어도 그 결과는 유효할 수 있다. 예를 들어 동영상을 감상하는데 경성 실시간처럼 마감 시간이 어겨진다고 해도 약간의 끊김 현상만 있을 뿐 시스템에는 별 영향이 없다. 즉, 연성 실시간 시스템에서도 역시 시간 제약성의 만족을 요구하지만 마감 시간이 어겨져도 결과가 치명적인 것은 아닌 경우를 말한다. 하지만 순수한 경성 실시간 시스템이나 연성 실시간 시스템이 모든 응용에 적합하지는 않다.

2.2 사례 연구

일반 목적의 운영 체제에서 실행되는 많은 웹 서버나 멀티미디어 컨텐트 서버의 경우 멀티미디어 어플리케이션, 또는 차별화된 서비스 요구에 의해 주어지는 주기성 또는 시간 제약성을 가지지만 경성 실시간 시스템과 같은 엄격함이 필요 없다. 시스템에 존재하는 시간 제약성이 없는 프로세스들도 적정한 서비스를 받아야 하기 때문에 프로세스의 시간 제약성과 대화성 사용자가 느끼는 프로세스의 응답의 중요성을 동시에 만족시키기 위한 통계적 보장이란 해결책이 제시되어 왔다.

리눅스와 같이 평균 성능 향상을 위해 동작하는 시분할 멀티태스킹을 하는 운영 체제 위에서 정확한 주기를 가지고 일을 해야 하는 프로세스에게 커널이 그 프로세스가 가지고 있는 시간 제약성을 엄격하게 맞춰준다는 것은 거의 불가능하다. 이를 개선하기 위한 여러 연구가 진행되어 왔다. 기존의 운영 체제를 수정하여 시간 제약성을 도모한 연구의 예는 RTAI[2], KURT[3], RTLinux[4], 비례 분할식 실행 방식[5], SMART[6], 리눅스 커널을 완전한 선점형 실시간 커널로 바꾸는 미국의 몬타비스타

사의 방식[7] 등 여러 가지가 있다.

2.2.1 RTAI, RTLinux

리눅스를 실시간 시스템으로 만들기 위한 노력인 RTAI[2], RTLinux[4] 기법에서는 리눅스 하부에 실시간 커널이 있어 그 스케줄러가 선점형 스케줄을 하며 리눅스 커널과 응용 프로그램을 그 실시간 커널 상의 한 비실시간 태스크로 실행한다. 실시간 커널의 스케줄러는 철저하게 우선 순위에 기반한 스케줄을 하기 때문에 낮은 지연시간 그리고 매우 예측이 가능한 시스템 환경을 제공한다. 또 정확한 프로세스 동작 시점을 유지하기 위하여, 커널 소스를 수정하여 인터럽트 처리 방법을 변경하였다. 이 방식에서는 주변 장치들로부터의 인터럽트를 실시간 커널이 가로채어 우선 순위가 높은 실시간 태스크가 우선 순위가 낮은 인터럽트에 의해 블록 되지 않도록 하며, 우선 수위에 따라 나중에 리눅스의 장치 드라이버의 인터럽트 서비스 루틴이 실행될 수 있도록 하고 있다.

RTAI, RTLinux 방식에서는 리눅스 상에서 실행되는 비실시간 태스크와는 달리 실시간 커널 위에서 실행되는 태스크는 모두 같은 메모리 공간에서 실행되어 메모리 보호 기능이 취약하며, 리눅스가 제공하는 서비스들을 이용할 수 없고, 실시간 비실시간 태스크들 사이에는 특별한 IPC (interprocess communication) 방법이 제공되어 그를 통해 서로 정보를 교환한다.

2.2.2 KURT

KURT[3]는 리눅스에서 태스크를 주기적으로 시행하기 위한 방법을 제공한다. KURT는 새로운 운영 체제가 아니고 어떠한 실시간 이벤트가 발생할 때 프로세스를 기동시켜 주는 스케줄링 기법이다. KURT는 일반적인 리눅스 프로세스들도 주기성을 가지고 실행될 수 있도록 만들기 때문에 RTLinux나 RTAI와 같은 환경에서 문제가 되는 응용 프로그램에서의 시스템 자원이나 기능의 사용 제한이 없다. KURT는 오로지 타이머를 조작하여 주기적인 프로세스 기동을 하기 때문에 인터럽트에 의한 프로세스의 기동 지연 문제를 해결하기 위하여 실제 기동 시점 이전에서부터 프로세스를 점유하는 비효율적인 방법을 사용하고 있다. 이 과정에 사용하는

UTIME 기법은 KURT 이외에서도 리눅스를 실시간화 연구에 많이 사용되고 있다. 또 KURT는 원천적으로 우선 순위에 기반한 스케줄을 하지 않기 때문에 완벽한 시간 제약성 만족을 보장하지 않고 통계적으로만 시간 제약성을 만족한다. 따라서 KURT는 경성이 아닌 실시간 환경에서 주기적인 데이터 수집 응용, 패킷을 주기적으로 전송하는 실험 장치 등을 구현할 때 리눅스가 제공하는 모든 기능을 그대로 활용할 수 있다는 측면에서 상당한 가치가 있다.

2.2.3 비례 분할식 실행 방식

비례 분할식 자원 할당은[5,8,9,10] 특히 일반적 용도의 운영 체제 내에서 실시간 서비스를 제공하는 문제에 잘 적용된다. 그 첫 번째 이유는 근본적인 스케줄링 메커니즘이 퀀텀에 기초한 스케줄러이며 두 번째 이유는 어떤 새로운 어플리케이션 차원의 개념이나 메커니즘을 도입하지 않고 비례 분할식 시스템을 구현할 수 있기 때문이다. 이것은 기존의 어플리케이션을 수정할 필요 없이 예측 가능한 실시간 방식으로 그대로 실행 가능하도록 만들 수 있음을 의미한다.

FreeBSD 운영 체제 하에서 비례 분할식 스케줄러를 구현한 연구[5]에서는 각 프로세스에 가중치를 할당하여 각 프로세스가 사용하는 CPU 자원의 양을 조절할 수 있도록 만들었다. 모든 프로세스는 기본적인 가중치를 할당받고 프로세스들은 시스템 호출에 의해 가중치를 변경할 수 있다. 스케줄러는 어떤 프로세스가 받아야 하는 CPU 자원의 할당량을 보장하기 위해 매번 프로세스가 추가되거나 가중치가 바뀔 때마다 주어진 자원 할당량을 유지하기 위한 가중치를 조정한다. 이러한 비례 분할식 스케줄 방식에서 인터럽트에 의한 네트워크 데이터의 수신, 수신된 데이터에 대한 프로토콜 처리 등은 스케줄러에 의해 제어되는 프로세스들보다 FreeBSD 운영 체제에서 더 높은 우선 순위를 가지고 실행된다. 이는 우선 순위가 낮은 프로세스를 위한 데이터가 우선 순위가 높은 프로세스 실행 중에도 처리될 가능성이 있다는 것을 의미하며 (우선 순위 역전 현상) 또 과도하게 많은 패킷이 지속적으로 수신되는 경우 인터럽트 처리와 프로토콜 처리를 위해 모든 CPU 자원이 사용됨으로써 우선 순위나 CPU 할당량과 상관없이 우선 순위가 높은 프로세스 조차 전혀 실행되지 못하는 상태가 만들어 질 수 있다 (라이브락 현상). 이 문제를 해결하기 위하여 [5]에서는

CPU 자원의 비례 할당 단위인 퀀텀이 종료하는 경우에만 네트워크 처리가 되도록 커널의 네트워크 데이터 처리 방법을 수정하였다. 이러한 방법은 일단 스케줄링 된 상태에서 할당된 퀀텀을 다 사용할 때까지 사용자 프로세스가 실행되는 것을 확실하게 만들어 주는 긍정적인 효과를 지니지만 커널 활동을 지연시키는 효과도 있다.

비례 분할식 CPU 할당 방식은 직관적으로는 주어진 가중치에 따라 CPU 자원을 배분하는 장점이 있으나 퀀텀, 즉 자원 배분의 최소 시간 단위의 정밀도가 자원 배분을 정확도를 결정하기 때문에 정밀도를 높이기 위하여 퀀텀을 짧게 잡은 경우 상당한 스케줄러 오버헤드가 생기게 된다. 또 네트워크 처리를 뒤로 미룸으로써 우선 순위가 높은 데이터의 경우에도 처리가 지연되는 또 다른 문제를 가지고 있다.
퀀텀 지속 시간은 시스템 성능에 중요한 영향을 미친다. 너무 길어도 안되고, 너무 짧아서도 안 된다.

2.2.4 SMART

Stanford SMART [6] 시스템은 멀티미디어 적용을 위한 실시간 스케줄러이고 Solaris 운영 체제에서 구현된다. 실시간 작업이 동시에 비실시간 태스크의 작업 반응을 받아들일 수 있는 수준을 제공하는 동안 마감 시간을 확실하게 하기 위해 weighted fair queuing 스케줄을 사용한다.
그것은 시스템 내부에 완전히 특색 있게 구현되고, 각종 운영 체제에 폭넓게 활용되기 때문에 생산 환경 내에서 쉽게 적응 할 수 있는 장점을 제공한다. 그러나 SMART는 주기적인 클럭을 사용하기 때문에 SMART가 제안하는 시간 결정을 제한한다.

3. Percentile 시스템

이 장에서 첫 번째로 우리는 일반적인 의미의 Percentile Goal을 정의한다. 따라서 우리는 큐 모델과 단순한 가정들을 기술할 수 있다. 정책수립을 명시해서 분석을 이끌러 내기는 어려우므로 우리는 제안 가능한 정책의 수립을 기술할 것이다.

3.1 정의

X_{ij}가 g_{ij} 시간 유닛의 실제 대기 시간을 감수할 수 있는 클래스 i 패킷의 부분이라고 정의하자. g_{ij}는 모든 i와 j에 대한 상수로 되었다. 그렇다면 Percentile Goal의 QoS 기준은 다음과 같이 표현되며

문제는 다음과 같은 식들을 보장할 수 있는 스케줄링 정책 f를 찾는 것이다:

$$
\begin{aligned}
& Wi\ (f) \quad g_{i1} \quad \text{for } X_{i1} \text{ fraction of packets,} \\
g_{i1} \quad & < Wi\ (f) \quad g_{i2} \quad \text{for } X_{i2} \text{ fraction of packets,} \\
& \vdots \\
& \vdots \\
g_i,\ M_{i-1} & < Wi\ (f) \quad g_{iMi} \quad \text{for } X_{iMi} \text{ fraction of packets,}
\end{aligned}
$$

물론 $X_{ij} + \cdot\cdot + X_{iMi} = 1$ 과 g_{ij}는 i와 j의 모든 값에 대한 상수이며 g_{iMi}는 아마도 무한대일 것이다. percentile goal 표준 분석의 첫 번째 단계는 미터법 적인 성능 측이며 우리는 클래스 i의 percentile goal인 M_i를 유한정수로 제한한다. 여기서 g_{ij} 가 사용자에 의해 구분된 상수인 점은 매우 중요하다. X_{ij}는 정책 f에 의해 스케줄된 함수이다. 이 장에서 우리는 2.3장에서 기술된 스케줄 정책에 대한 취득할 수 있는 모든 X_{ij}의 수립을 찾기를 원할 것이다. 이 장의 두 번째 파트에서 우리는 정책 f를 수립할 것이다. 지금, 우리는 위에서 언급한 B-ISDN 링크 상의 음성, 비디오 등 Data 전송의 문제점을 통해 모델화 시킬 수 있다. 우리가 표시한 Data, 음성, 비디오의 등급을 1, 2, 3으로 구분한다고 가정하자. 클래스 1의 패킷은 별로 지연현상에 민감하지 않고 아마도 10초 정도의 큰 지연도 감수할 수 있다. 클래스 2의 패킷은 지연에 매우 민감하며 그들의 93%는 25msec 이상의

지연을 경험하면 안될 것이다. 클래스 3의 패킷은 그들에게 많은 종류의 지연이 있기 때문에 지연에 그다지 민감하지 않을 것이다. 따라서 우리가 200ms의 상대적으로 큰 지연을 감수하더라도 지연이 대부분 패킷에 해당하는 210ms 보다 클 수는 없다. 이 문제는 percentile goal criterion의 정의를 다음과 같이 정해 재구성할 수 있다:

$W_1 \leq 10$ for 100% packets, $W_2 \leq .025$ for 95% packets,
$W_2 > .025$ for 5% packets, $W_3 \leq .2$ for 5% packets,
$W_3 < .21$ for 90% packets $W_3 > .21$ for 2% packets

실제로 일 부분의 패킷 수신의 매우 큰 지연도 감수할 수 있다. 예를 들어, 위의 예제 클래스 3의 2% 패킷은 비제한적 지연 경험을 허용한다는 뜻이다.
참고 문헌 [1]에 이 문제에 관한 수학적 모델과 해가 있다.

4. 리눅스 Percentile Scheduler

4.1 Process Scheduler에 Percentile 적용

Percentile 스케줄링 기법은 패킷 별 우선 순위에 기반한 네트워크 라우터에 사용된 스케줄링 기법이다[1]. 이 기법에서는 라우터에 도착하는 패킷들을 서비스 품질에 따른 클래스로 구분하고, 각 클래스에 최대 지연 시간과 그 클래스에 속하는 패킷들이 정해진 지연 시간 이내에 전송되는 비율 즉 Percentile 을 정하고 그 비율을 만족시키기 위한 스케줄링 방법이다. Percentile 스케줄에서 클래스 i 에 속하는 패킷의 스케줄 우선 순위는 다음과 같다.

$$\Pr(i) = \frac{Si}{Ni * P} \qquad (1)$$

식에서 Si 는 주어진 마감 시간 안에 처리된 패킷의 개수이며 Ni 는 현재까지 클래스 i 로 분류된 패킷의 개수, P 는 주어진 Percentile 이다. 이 $\Pr(i)$ 값이 낮을수록 높은 우선 순위를 가지게 된다. 즉 마감시간 안에 처리된 패킷의 수가 Percentile에 비하여 상대적으로 적은 클래스의 패킷이 더 우선하여 처리된다는 것이다. 이 식의 유도 과정은 [1]에서 볼 수 있다.

패킷 전송에 사용되는 Percentile 스케줄 방식은 한 패킷의 최대 길이가 제한되어 있고 패킷이 전송되고 있는 동안에는 다른 패킷이 중간에 끼여들 수 없다는 점이 가정된다. 반면에 비 주기적으로 실행되는 일반적인 프로세스 스케줄의 경우에는 한 태스크의 실행 시간을 특정 값으로 한정하는 것이 적당하지 않기 때문에 프로세스의 실행 시간과 스케줄 단위에 관한 특별한 고려가 필요하다. 이 고려 사항은 실제 스케줄러가 구현되는 응용 환경에 따라 각각 정의될 수 있으며 이 논문에서는 서버에 도착하는 각 페이지 요구 (http 서버 입장에서의 하나의 페이지 요구)는 하나의 태스크로 간주하고 모든 태스크는 같은 실행 시간과 마감 시간을 가진다고 가정하였다.

4.2 리눅스에서의 구현

이 논문에서는 리눅스의 스케쥴러를 Percentile에 기반한 스케쥴러로 바꾸고자 한다. 따라서 먼저 리눅스의 스케쥴러가 어떤 방법으로 스케쥴을 하는지 살펴볼 필요가 있다.

4.2.1 원래의 리눅스 스케줄러

유닉스 운영체제의 스케쥴링 알고리즘은 몇 가지 서로 상충되는 조건을 갖춰야한다. 프로세스 반응 시간이 짧아야 하고, 후면 작업도 효율적으로 처리해야 한다. 그렇다고 CPU를 사용하지 못하는 프로세스가 있어서도 안되며, 낮은 우선 순위의 프로세스와 높은 우선 순위의 프로세스를 중재할 수도 있어야 하며, 기타 여러 사항을 고려해야 한다.
리눅스의 스케쥴링은 시분할 기법은 토대로 한다. 즉 CPU 시간을 '슬라이스(slice)'로 쪼개고, 실행 가능한 각 프로세스마다 슬라이스를 하나씩 할당하여 프로세스 여러 개를 동시에 실행한다. 물론 프로세서 하나는 동시에 프로세스 하나만 실행할 수 있다. 현재 실행하고 있는 프로세스를 종료하지 않았는데, 프로세스에게 주어진 타임 슬라이스 또는 타임 퀀텀이 완료되면 프로세스 전환이 일어난다. 시분할은 타이머 인터럽트에 의존하기 때문에 프로세스는 자연히 시분할에 대해 신경 쓸 필요가 없다.

프로세스의 우선 순위에 따라 프로세스들의 순위를 매기는 것 또한 스케쥴링 정책의 기반이 된다. 프로세스의 현재 우선 순위를 얻어 오는 데는 때론 복잡한 알고리즘을 사용하기도 하지만, 그 최종 결과물은 똑같다. 이는 각 프로세스가 CPU를 할당받는데 어느 정도 적합한가를 나타내는 값이다.
리눅스에서 프로세스의 우선 순위는 동적으로 매겨진다. 스케쥴러는 프로세스를 계속 지켜보면서 주기적으로 프로세스의 우선 순위를 조정한다. 스케쥴러는 오랜 시간동안 CPU를 사용하지 못한 프로세스의 우선 순위를 동적으로 높인다. 이와 마찬가지로 오랫동안 CPU를 사용한 프로세스의 우선 순위를 동적으로 낮춘다.
일반적으로 스케쥴링을 이야기할 때 프로세스를 '입출력 위주'와 'CPU 위주'로 분류한다. 전자는 입출력 장치를 많이 이용하고, 많은 시간을 입출

력 작업이 끝나기를 기다리는 데 사용한다. 후자는 숫자 계산을 하는 프로그램처럼 CPU 시간이 많이 필요한 응용프로그램이다.

리눅스 프로세스는 선점형 (preemptive)이다. 만약 어떤 프로세스가 TASK_RUNNING 상태가 되면 커널은 이 프로세스의 동적 우선 순위가 현재 실행 중인 프로세스의 우선 순위보다 높은지 검사한다. 만약 그렇다면 현재 실행중인 프로세스의 실행을 중단하고, 스케쥴러를 호출하여 다른 프로세스를 선택해서 실행하도록 한다. 물론 프로세스는 자신에게 주어진 타임 퀀텀을 모두 사용했을 때에도 선점될 수 있다. 시분할 방식에서 이런 일이 벌어지면 현재 프로세스가 스케쥴이 필요하다는 플래그를 켠다. 그러면 타이머 인터럽트 핸들러가 끝날 때쯤에 스케쥴러를 호출한다.
예를 들어. 두 프로그램(예를 들어, 문서 편집기와 컴파일러)만 실행되고 있는 상황을 가정하자. 문서 편집기는 상호 작용이 가능한 프로그램이다. 따라서 문서 편집기는 컴파일러보다는 동적 우선 순위가 높다. 그렇지만 사용자는 생각하기 위한 휴식과 데이터 입력을 번갈아 수행하고, 게다가 키 두 개를 입력하는 사이의 평균 지연시간이 상대적으로 길기 때문에 이 프로그램은 자주 보류된다. 그러나 사용자가 키를 입력하자마자 인터럽트가 발생하고, 커널은 문서 편집기 프로세스를 깨우게 된다. 커널은 현재 실행 중인 프로세스(컴파일러), 현재 실행중인 프로세스의 우선 순위 보다 문서 편집기의 동적 우선 순위가 높다고 판단하기 때문에 현재 프로세스가 스케쥴이 필요하다는 플래그를 켜고, 커널이 인터럽트 핸들러를 마칠 때 강제로 스케쥴러를 호출한다. 스케쥴러는 문서 편집기를 선택하고 작업 전환을 수행한다. 이 결과로 문서 편집기는 실행 상태로 빠르게 복귀하여 사용자가 입력한 글자를 화면에 보여준다. 입력한 글자를 처리한 후 문서 편집기 프로세스는 또 다른 키 입력을 기다리며 보류 상태가 되고, 컴파일러 프로세스를 다시 실행한다.

리눅스의 스케쥴링 알고리즘은 시기(epoch) 단위로 CPU 시간을 나누어 동작한다. 단일 시기에서, 모든 프로세스는 시기가 시작할 때 계산한 주기만큼의 지정된 타임 퀀텀 값을 가진다 일반적으로 각 프로세스마다 서로 다른 타임 퀀텀 지속 시간을 준다. 타임 퀀텀 값은 그 시기 동안 프로세스에 할당되는 최대 CPU 시간이다. 타임 퀀텀을 모두 소비하고 나면 프로세스는 선점되며, 실행할 수 있는 다른 프로세스로 대체된다. 물론 프로세

스가 퀀텀을 모두 소비하지 않는 한, 같은 시기에 스케줄러가 여러 번 프로세스를 선택하여 실행할 수도 있다. 예를 들어, 입출력을 완료하기를 기다리기 위해 프로세스 수행을 보류하면 자신에게 남은 타임 퀀텀을 보존하고 있다가 같은 시기에 다시 선택될 수 있다.
한 시기는 실행 가능한 모든 프로세스가 자신의 퀀텀을 모두 소비했을 때 끝난다. 이 경우 스케줄러 알고리즘은 모든 프로세스의 타임 퀀텀을 다시 계산하고, 새로운 시기를 시작한다.
각 프로세스에는 기본 타임 퀀텀이 있다. 이는 프로세스가 이전 시기에 자신의 퀀텀을 모두 소비했을 때 스케줄러가 프로세스에 할당하는 타임 퀀텀 값이다. 사용자는 자신이 실행한 프로세스의 기본 타임 퀀텀을 nice()와 setprioity() 시스템 콜을 이용하여 바꿀 수 있다.
스케줄러는 각 프로세스의 우선 순위를 고려하여 실행할 프로세스를 선택한다. 실제로 리눅스에는 정적 우선 순위(static priority)와 동적 우선 순위(dynamic priority) 두 가지 종류의 우선 순위가 있다.
정적 우선 순위는 사용자가 프로세스에게 부여한 것으로 1부터 99까지의 값이다. 이 값은 스케줄러로 바꿀 수 없다. 동적 우선 순위는 일반 프로세스에만 적용하며, 본래는 기본 타임 퀀텀과 현재 시기에서 자신의 퀀텀이 만료되기까지 프로세스에 남은 CPU 시간 동안 틱 횟수의 합이다.

원래의 리눅스는 스케줄러의 시스템 호출을 통해 선택될 수 있는 FIFO (first in first out), RR (round robin), OTHER 방식의 세 가지의 스케줄링 클래스를 유지한다. 리눅스 스케줄러는 FIFO, RR, OTHER의 순서로 프로세스를 우선 스케줄 한다.
FIFO 방식은 먼저 들어온 것이 먼저 나가는 방식의 프로세스로서 스케줄러가 프로세스에게 CPU를 할당할 때 실행 큐 리스트에서 현재 위치에 있는 프로세스를 가져온다. 우선 순위가 더 높은 실행 가능한 실시가 프로세스가 없다면, 우선 순위가 같은 다른 프로세스가 있다고 하더라도 자신이 원한다면 계속해서 CPU를 사용할 수 있다.
RR 방식은 스케줄러가 프로세스에 CPU를 할당할 때 선택한 프로세스를 실행 큐 리스트의 맨 끝에 넣는다. 이 정책은 우선 순위가 같은 프로세스 사이에서 공정한 CPU 시간 할당을 보장한다.
OTHER 방식은 일반 시분할 프로세스로서 CPU 시간을 '슬라이스(slice)'로 쪼개고, 실행 가능한 각 프로세스마다 슬라이스를 하나씩 할당하여 프

로세스 여러 개를 동시에 실행한다. 물론 프로세서 하나는 동시에 프로세스 하나만 실행할 수 있다. 현재 실행하고 있는 프로세스를 종료하지 않았는데, 프로세스에게 주어진 타임 슬라이스 또는 타임 퀀텀이 완료되면 프로세스 전환이 일어난다.

리눅스 스케줄러는 매번 스케줄러에 의해 계산된 가중치에 기반한 스케줄을 한다. 스케줄러의 가중치 계산에는 스케줄 클래스, CPU 점유 시간, 시스템 호출에 의해 결정되는 nice 값, 캐쉬와 멀티 프로세스를 고려한 CPU 친근도 등이 고려된다. 리눅스는 자신에게 할당된 퀀텀을 다 사용하거나 프로세스 자체가 입출력이나 다른 시스템 호출에 의하여 블록 되는 경우를 제외하고는 중단되지 않는다. 따라서 모든 프로세스가 가중치에 기반 하여 스케줄링 되기 때문에 데드락 상태에 빠지지 않는 이상 모든 프로세스가 비교적 공정하게 CPU 서비스를 받게 된다. 이 말은 리눅스가 철저한 우선 순위에 의한 서비스 제공을 바탕으로 한 실시간 응용에 적합하지 않다는 것을 의미한다. 그럼에도 불구하고 상당히 많은 웹 서버나 멀티미디어 서비스가 리눅스 상에서 이루어지고 있는 것은 CPU의 성능에 의존한 빠른 응답을 할 수 있기 때문이다. 하지만 이 방법은 서비스 요구의 증가에 따라 시스템 사용량이 100%에 가까워지면 질수록 적절한 응답 시간을 어떤 요구에 대해서도 보장할 수 없는 문제가 생긴다. 따라서 서비스 요구가 아주 많은 상황에서도 적정 수준의 서비스 응답 시간을 통계적으로 보장하고, 오버로드 상황에서도 차별된 서비스를 일부 요구에 대하여 제공할 수 있는 방법이 필요하다.

4.2.2 우리가 구현한 스케줄러

우리는 두 가지 레벨의 서비스 품질이 존재하는 경우를 연구한다. 예를 들어, 호스팅 서비스는 호스팅 비용을 지불하는 기업 고객을 위해서는 고품질의 서비스를, 무료로 서비스를 받는 개인 고객을 위해서는 낮은 수준의 서비스를 제공할지 모른다. 단순하긴 하지만, 그 문제는 차별화된 QoS 메커니즘을 위한 좋은 시범 케이스를 제공한다. 우리는 차별화된 품질의 서비스를 제공하기 위한 우선 순위 기반 접근법을 조사한다. 즉, 모든 서비스 요구에는 우선 순위가 부여되고 그 우선 순위에 따라 서비스된다. 명백히, 모든 서비스 요구에는 그것이 어디로부터 오는지가 아닌, 어떤 문서

를 요구하고 있는지를 기반으로 우선 순위가 할당된다. 다시 말하면, 고품질의 서비스를 구매하는 고객들은 누가 문서에 액세스하고 있는지에 관계없이, 자신의 as서에 대한 웹 액세스가 빠르기를 원한다.
현재의 웹 서버에서는 우선 순위의 관점에서 보면 수신된 요구들 사이에 어떤 차이가 없으므로, 우리는 웹서버를 수정하기 위한 두 접근법을 사용하였다:

• 사용자-레벨 접근 : 대중적 웹 서버 프로그램인 아파치 서버를 수정하여 수신된 요구가 서비스되어야 하는 우선 순위를 결정하는 코드를 첨가하였다.

• 커널-레벨 접근 : 커널을 수정하여 수신된 요구의 우선 순위로부터 프로세스 우선 순위로의 사상(mapping)을 제공하고, 어느 우선 순위 레벨에서 어떤 프로세스가 실행되고 있는지 추적하는 새로운 시스템 호출들을 추가하였다.

스케줄링 정책은 수신된 요구가 서비스되어야 하는 순서를 결정한다. 스케줄링 정책은 선점권(preemptive)이 있거나 비선점적(non-preemptive)일 수 있다. 사용자 레벨에서는 새로운 프로세스의 실행을 허용하기 위해 실행중인 프로세스를 인터럽트 하거나 차단할 수 없기 때문에, 우리는 커널-레벨에서의 선점적 스케줄링을 구현하였다.
스케줄링 정책에는 두 가지 중요한 측면이 존재한다. 첫 번째, 요구를 수신하자마자, 스케줄링 정책은 그 요구를 즉시 프로세싱 할 것인지 아니면 실행을 연기할 것인지를 결정해야 한다(sleep policy). 수신된 요구는 다음의 이유들 중 하나로 인해 연기될 수 있다.

• 시스템 부하가 이미 높다면, 현재 수신된 요구를 실행하는 대기 시간이 영향을 받을 것이다.

• 수신된 요구가 낮은 우선 순위에 있다면, 이후에 도착하는 높은 우선 순위의 요구가 영향을 받을 수 있다.

두 번째, 스케줄링 정책은 또한, 연기되었던 요구의 계속 허용되어야 하

는 때를 결정해야 한다(Wakeup policy). 우리는 완료된 요구를 대신할 때에만 요구를 허용한다. 수신된 요구가 완료될 때, 스케줄링 정책은 , 만약 있다면, 연기된 요구들 중 어느 것이 그 대신으로 수행되도록 선택되어야 할지를 결정해야 한다.

그림 5는 Percentile에 기반한 스케줄러의 구성도 이다. 먼저 수신된 요구는 클래스를 부여받기 위한 대기 큐 (unresolved task queue)에 들어가며 CPU가 더 이상 실행할 태스크가 없는 경우에 이 요구를 분류하여 클래스 *0*부터 클래스 *N-1* 까지 *N* 개의 클래스로 분류하여 각 클래스의 큐에 대기시킨다. 스케줄러는 위의 우선 순위 계산식 (1)에 기반 하여 가장 우선 순위가 큰 요구를 꺼내어 스케줄 한다. 이 과정에서 실제로는 위 우선 순위 계산식을 적용한 결과, 발생할 수 있는 아주 작은 우선 순위 차이를 무시하고 스케줄 대상이 될 후보 집합을 만든 뒤에 그 집합에 있는 태스크 가운데 EDF (Earliest Deadline First) 기준에 따라 가장 먼저 도착한 (모든 태스크의 마감 시간이 일정하다고 가정하였으므로) 태스크를 골라 수행한다. 우리가 사용한 서버는 실시간 시스템이 아닌 일반적인 서버로서 이러한 방법은 시스템의 사용도가 아주 높을 때 대화형 사용자(주로 X 윈도우 기반 응용 프로그램을 사용하는)의 응답 시간을 매우 길게 할 수 있는 단점이 있다. 따라서 CPU가 실행하는 다른 프로세스와 대화형 프로세스를 위하여 CPU 대역폭의 1% 정도를 할당하고 나머지 99%의 CPU 대역폭은 서버에 들어오는 클라이언트의 요구 처리를 위하여 사용한다.

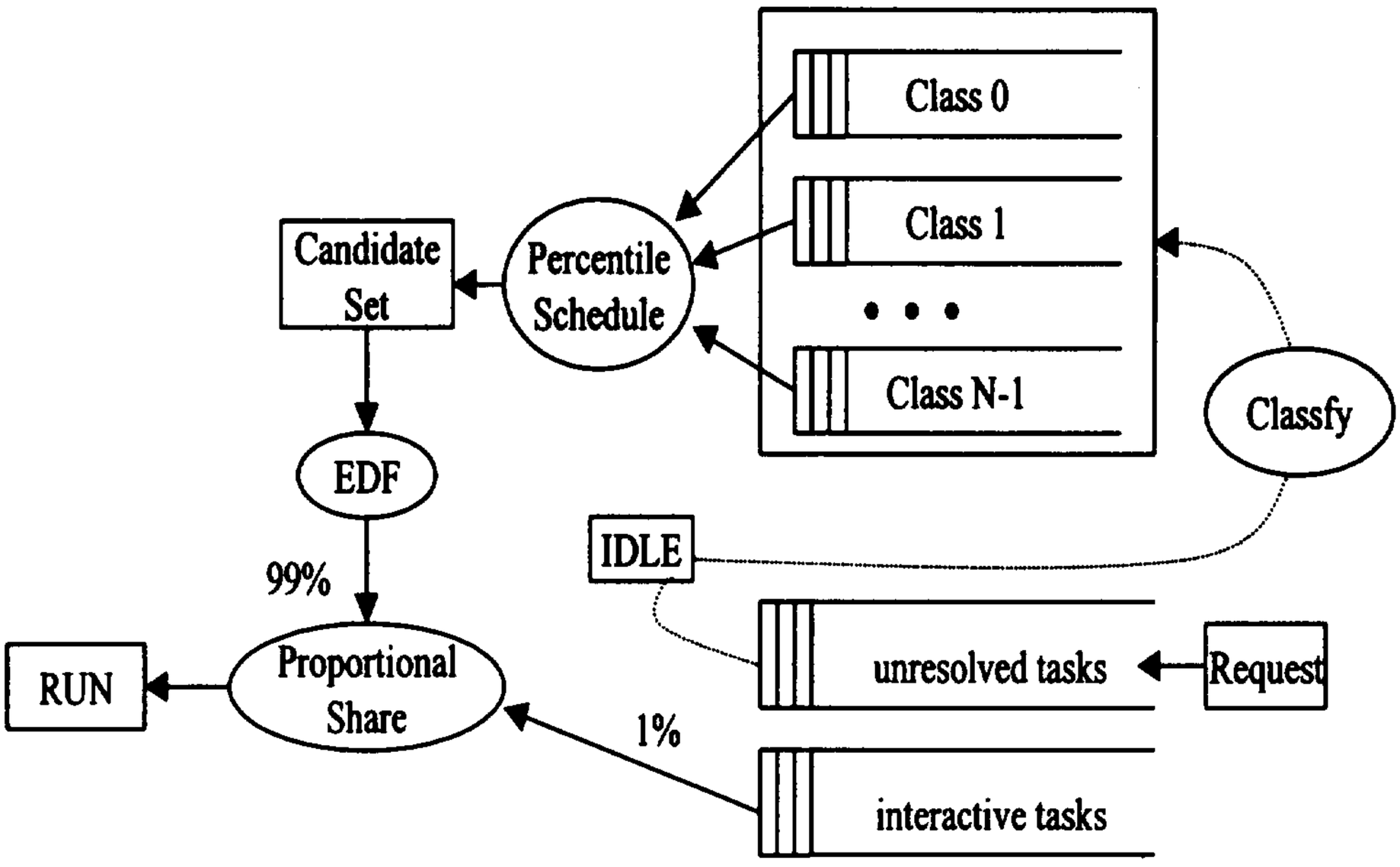

그림 1 Percentile 스케줄러의 구성

5. 실험 및 결과 분석

5.1 실험 대상 및 환경

5.1.1 웹 서버

클라이언트 시스템에는 페이지 요구를 지속적으로 발생시키는 가상적인 브라우저 프로그램이 실행되며 실제 페이지 요구는 한 이더넷 패킷에 담을 수 있는 작은 정적인 페이지에 대하여 이루어졌다.

사용된 서버에는 앞 절에서 제시한 스케쥴러가 구현되었으며, 특히 실험에서 사용된 과도한 페이지 요구를 처리하기 위하여 최대 프로세스 수 (task.h의 NR_TASK 상수), 최대 소켓 수 (socket.h의 SOMAXCONN 상수) 등의 값을 크게 조정하였다. 클라이언트의 경우 TCP를 이용한 페이지 요구와 응답의 경우 서버의 부하가 증가할 때 응답 시간이 길어져 단위 시간 당 페이지 요구의 수를 크게 높일 수 없는 문제가 있어, 실험에서는 클라이언트가 서버에게 연결을 요청하거나 페이지 요구를 한 뒤에 50msec 이내에 응답이 없는 경우 현재 진행되는 연결을 리셋하고 다시 연결을 요청하는 방법을 사용하였다.

5.1.2 실험 환경

실험은 리눅스가 실행되는 서버 PC와 페이지 요구를 발생하기 위한 클라이언트 PC를 100Mbps 스위치 기반 이더넷으로 구성되었다. 서버는 256M 바이트의 메인 메모리를 가진 Pentium II 400MHz의 PC이며 리눅스에 널리 사용되는 아파치 서버가 http 서버로 사용되었다. 클라이언트는 동급의 PC 두 대와 SUN OS 166MHz Ultra Sparc 컴퓨터 두 대가 사용되었으며 이 네 대의 클라이언트가 끊임없이 서버에 패킷을 요구하도록 실험 환경을 만들었다.

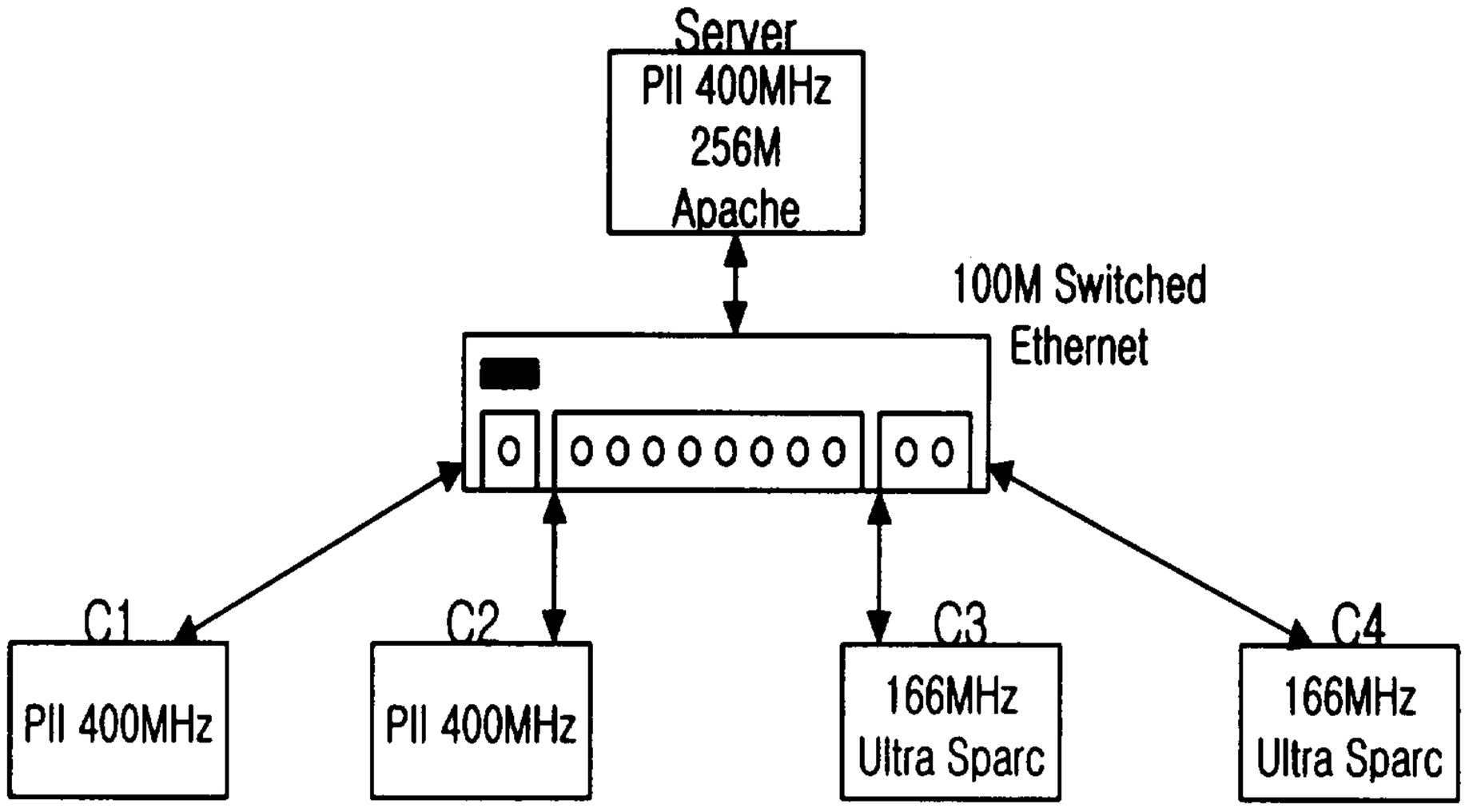

그림 2 실험 환경

5.1.3 실험 방법

실험에서 서버는 들어오는 페이지 요구를 다섯 개의 클래스로 분류하고 각 클래스에 대하여 50%에서 90% 까지 10% 단위로 Percentile을 할당하였다. 패킷이 수신되면 각 패킷의 클래스에 할당된 Percentile에 기반한 우선 순위에 의거하여 스케줄된다. 각 페이지 요구는 1msec 내에 CPU에 의해 처리된다고 가정하였으며 이 1msec를 마감 시간으로 정하였다. 즉 각 클래스의 큐에 들어온 뒤에 1msec 이내에 요구가 처리되면 마감 시간을 만족한 것으로, 1msec 이상이 걸리면 스케줄러는 이 클래스의 요구에 대하여 마감 시간을 지키지 못한 것으로 간주하였다.

5.2 실험 결과

5.2.1 일반적 결과

실험에서는 위 실험 환경에서 Percentile 스케줄을 적용한 경우와 적용하지 않은 경우 클라이언트의 페이지 요구에 대한 응답 시간을 비교하였다. 먼저 그림 7은 페이지 요구를 다섯 개의 클래스로 분류하여 차등적인

서비스를 한 결과 클라이언트에서 관찰된 응답 시간을 그래프로 표현한 것이다. 이 응답 시간은 클라이언트 시스템에서 관찰된 시간이기 때문에 서버로의 TCP 연결 요구를 한 시점부터, 페이지 요청, 결과 수신까지의 모든 시간을 더한 것으로 서버가 관찰하는 시간과는 다른 시간이다. 따라서 80 ms 정도에 페이지 요구를 모두 처리한 것을 볼 수 있다. 그림에서 X 축은 클라이언트에서 관찰된 응답 시간이며 Y 축은 누적된 패킷 비율이다. 그림에서 Percentile이 높을수록 응답 시간이 빠르다는 것을 확연히 볼 수 있다.

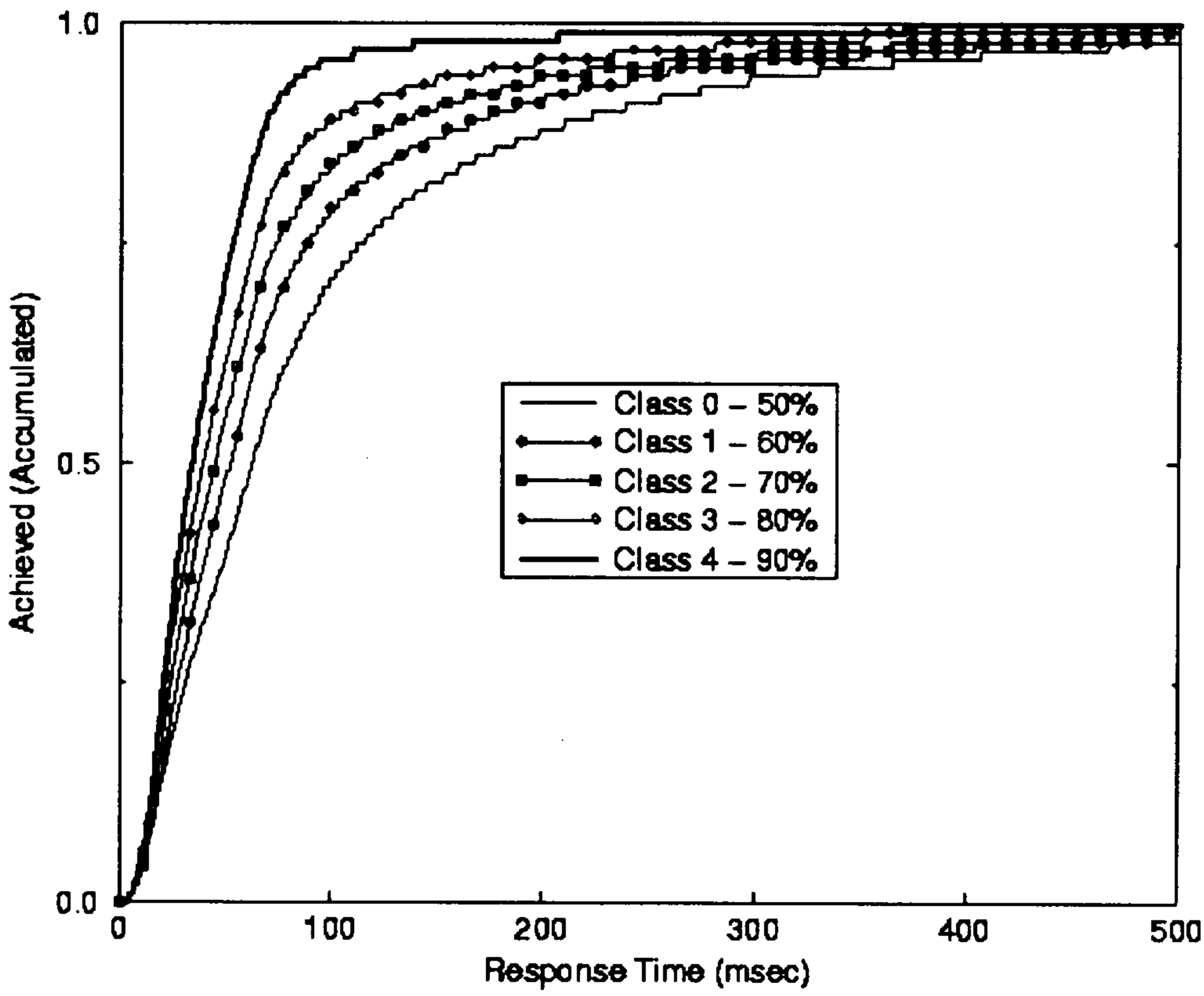

그림 3 Percentile 스케줄 결과

그림 8은 같은 실험 환경에서 http 서버의 수정 없이, 또 원래의 리눅스 스케줄러를 그대로 이용하여 요구를 처리한 경우의 그래프이다. 원래의 리눅스는 모든 프로세스가 가중치에 기반 하여 스케줄링 되기 때문에 데드

락 상태에 빠지지 않는 이상 모든 프로세스가 비교적 공정하게 CPU 서비스를 받게 된다. 그러므로, 이 경우에는 각 클래스 별 우선 순위가 하나도 적용되지 않았기 때문에 모든 클래스에 대하여 같은 응답 시간을 보이고 있다.

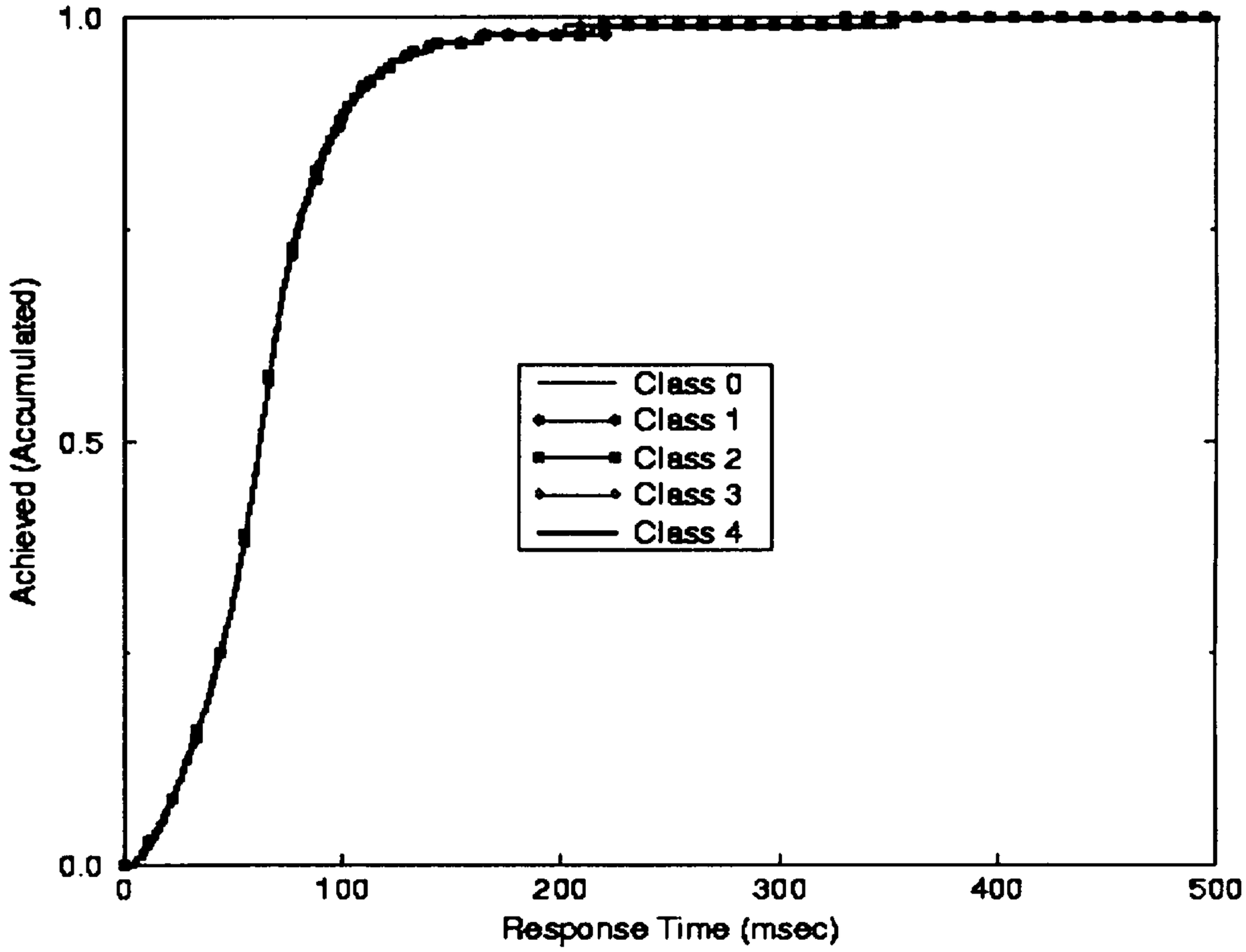

그림 4 시분할 스케줄 결과

5.2.2 드러난 문제

그림 9는 리눅스에서 네트워크 데이터의 처리 과정을 보이고 있다. 그림에서처럼 네트워크 인터럽트가 걸려 패킷이 수신되면 실제 프로토콜 처리는 인터럽트에서 리턴한 뒤에 운영 체제의 네트워크 하반부 (network bottom half)에서 처리된다. 네트워크 프로세싱 결과 추출된 요구에는 http 서버가 스케줄 되어 수신한 이후에야 우선 순위가 할당된다. 따라서 패킷의 수신 후에 우선 순위가 할당되기까지는 모든 클래스에 대한 요구와 다

른 네트워크 데이터에 대하여 모두 같은 우선 순위가 적용되어 도착순으로 처리된다는 것을 의미하며, 과도한 패킷이 수신되어 버퍼 부족으로 인하여 패킷이 버려질 때에도 http 서버가 분류하는 우선 순위와는 상관없이 버려질 패킷이 선택된다. 즉 서버에 의해 우선 순위가 결정되기까지는 우선 순위의 무시 내지는 역전을 겪으면서 스케줄이 되어 완전한 우선 순위 기반 스케줄 결과와는 약간 차이를 보이게 된다. 다시 말해 좀더 무딘 결과를 얻게 된다.

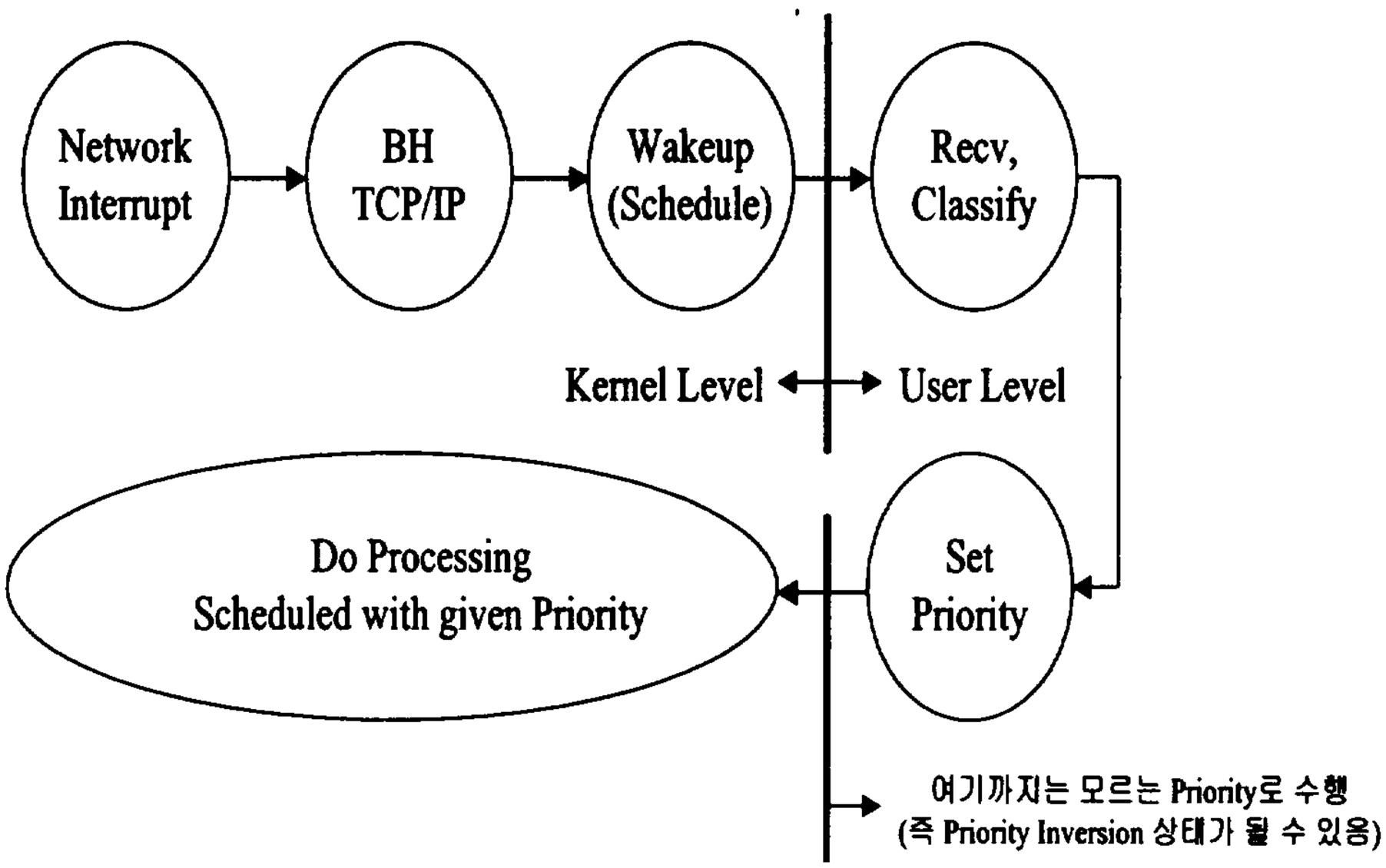

그림 5 리눅스에서의 네트워크 데이터 처리

6. 결론

이 논문에서는 패킷 스케쥴에 사용된 Percentile 스케쥴 기법을 웹 서버에 적용하여 서버에 들어오는 다양한 요구를 클래스로 분류하여 차등적인 서비스를 제공하고자 하는 연구를 기술하였다. 논문에서는 범용 운영 체제를 실시간화 하기 위한 노력들을 살펴보았으며, Percentile에 기반한 스케쥴러를 리눅스 환경에 구성한 방법을 기술하였다. 제안된 기법의 성능을 살펴보기 위하여 실험을 수행하였으며 Percentile 스케쥴 기법을 적용한 경우 지정된 클래스에 따라 확연히 다른 응답 시간을 관찰할 수 있었다. Percentile 스케쥴 기법은 기존의 시분할 스케쥴로는 할 수 없는 새로운 서비스를 가능하게 하며, 이를 바탕으로 서버가 사용되는 많은 시스템에서 더 많은 수익을 창출할 수 있는 기회를 제공할 수 있다.

예를 들어, 서버 호스팅의 경우, 서로 다른 비용을 지불한 호스팅 고객에 대하여 같은 수준의 서비스를 제공하는 것은 공정하지 못하다. 많은 비용을 지불한 고객의 홈페이지에 대한 참조에 비하여 적은 비용을 지불한 고객의 홈페이지 보다 더 짧은 응답을 제공하여 호스팅 업체는 같은 서버 안에서 차별화 된 서비스를 할 수 있게 된다. 또 전자 상거래 사이트의 경우, 사이트를 단순히 살펴보기 위하여 방문한 단발성 고객과 자주 물건을 구매한 고객, 또 그들 중에서도 물건을 구매하기 위하여 쇼핑 카트에 상품을 넣은 고객 등에 대하여 다른 품질의 서비스를 제공함으로써 더 많은 구매를 유도할 수 있다. 즉, 웹 호스팅 업체나 전자 상거래 사이트 등에서 더 많은 수익을 창출할 수 있는 기회를 제공한다.

이 연구의 후속 연구로는 스케쥴러를 더 개선하여 프로토콜 프로세싱 과정에서도 우선 순위를 지정하여 확실한 우선 순위 정책을 구현하는 일과, 하드디스크, 데이터 베이스 시스템 등 일반적으로 서버에서 이용되는 다른 서비스가 연계되었을 경우에도 클래스 별 우선 순위를 지정할 수 있도록 만드는 일이다.

참고 문헌

[1] Agarwal, N. "Percentile Goal As A Performance Criterion In A Multiclass GI/G/1 Queuing System", Ph.D. thesis, North Carolina State University, Raleigh, NC, USA, 1995.

[2] http://www.aero.polimi.it/projects/rtai/

[3] http://www.ittc.ku.edu/kurt/

[4] http//www.rtlinux.org

[5] K. Jeffay, F. Donelson Smith, A. Moorthy, J., Anderson, Proportional Share Scheduling of Operating System Services for Real-Time Application", proceeding of the 19th IEEE Real-Time Systems Symposium, Dec. 1998, pp. 480-491.

[6] J. Nieh, M.S. Lam, The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications, Proc., 16th ACM Symp. on Operating System Principles, Saint-Malo, France, Oct. 1997, pp. 184-197.

[7] http://www.hardhatlinux.com

[8] P. Goyal, X. Guo, H. M. Vin, A Hierarchical CPU Scheduler for Multimedia Operating Systems, Proc., USENIX Symp. On Operating Systems Design and Implementation, Seattle, WA, Oct, 1996, pp. 107-121.

[9] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. Baruah, J. gehrke, C. Plaxton, A Proportional Share Resource Allocation Algorithm for Real-Time, Time-Shared Systems, Proc. 17th IEEE Real-Time Systems Symposium, Dec. 1996, pp. 288-299.

[10] C. Waldspurger, W. Weihl. Lottery Scheduling: Flexible Proportional Share Resource Management, Proc. USENIX Symp. On Operating System Design and Implementation, Nov. 1994, pp. 1-12.

ABSTRACT

An Implementation of Percentile Scheduler on Linux

Choi, Dong-Joon
Computer New Technology
of the Graduated School
Hansung University

Applications such as interactive multimedia and virtual environments, require real-time computation and communication services from the operating system in order to be effective. As application with various time constraints are increasingly being hosted on general purpose (rather than specialized real-time) operating systems, the great interest in migrating real-time systems technology to desktop operating systems grows. In this paper we implemented a percentile-based web server to provide differentiated services for the different classes of service requests. By implementing percentile based scheduler and modified web server on Linux operating system, we

showed that our scheduling mechanism clearly differentiate services based on the QoS requirements defined as percentile.