論文
6-3-2

클러스터 웹서버 성능개선을 위한 과부하 방지 QoS 보장 수락 제어 기법 연구

(Study on An Admission Control Scheme to Improve QoS of Overload Cluster Web Server)

남의석(Eui-seok Nahm)[1] 강이구(E.G. Kang)[2] 정헌석(H.S. Chung)[3] 이준환(J.H. Lee)[4] 현득창(D.C. Hyun)[5]

국문요약

최근 몇 년 사이 컴퓨터와 초고속 통신망의 대중적인 보급으로 인하여 인터넷 사용 인구가 급격하게 증가하였고 인터넷 사용량 또한 폭발적으로 증가하였다. 이와 같이 막대한 양으로 늘어난 트래픽은 서버에 과도한 양의 부하를 발생시켜 서비스 전반에 걸친 성능 저하를 가져왔다. 따라서 서버에 많은 양의 트래픽이 발생하더라도 정상적인 서비스를 제공할 수 있도록 하는 것이 필요하다.

본 논문에서는 이러한 서버의 트래픽 폭주로 인한 서버 과부하 문제를 해결하기 위해 수락 제어 알고리즘을 이용한다. 수락 제어 기법은 컨텐츠 변환이나 로드 밸런싱과 같이 수동적으로 부하를 덜어주거나 분산시키는 방식이 아닌 트래픽에 제한을 가하여 능동적으로 서버의 부하를 조절하는 방식이다. 클라이언트에 의한 연결 요청이 이루어질 때 서버의 부하 상태를 평가한 후 서버가 수용 가능한 수준이면 클라이언트 새로운 연결 요청을 수락하여 이를 서비스할 수 있도록 한다. <중략>

또한 서비스 요청에 제한적인 정책을 따를 경우 서버의 부하 변화에 빠르게 반응하지만 QoS(Quality of Service)의 향상을 얻을 수 있는 반면 더 많은 서비스 요청을 수용하는 정책을 취할 경우 서버의 부하 변화에 대한 반응이 느리지만 throughput이 증가하는 이득을 얻을 수 있게 된다. QoS와 throughput가 서로 tradeoff 관계에 있다는 것을 알 수 있다. 또한 요구되는 서비스의 성격에 따라 즉 더 높은 수준의 QoS 보장이 중요한 서비스인가 아니면 더 높은 수준의 throughput 보장이 요구되는 서비스인가에 따라 다른 수락 제어 정책을 적용하는 것이 가능하게 된다.

논문접수 : 2005. 6. 25.
심사완료 : 2005. 7. 25.

1) 정회원 : 극동대학교 컴퓨터정보표준학부
2) 정회원 : 극동대학교 컴퓨터정보표준학부
3) 정회원 : 극동대학교 컴퓨터정보표준학부
4) 정회원 : 극동대학교 컴퓨터정보표준학부
5) 정회원 : 극동대학교 컴퓨터정보표준학부

본 논문은 한국표준협회의 지원에 의해 쓰여진 논문임.

1. 서 론

최근 몇 년 사이 고속 인터넷의 대중적인 보급 및 관련 인프라의 확충으로 인하여 인터넷 사용 인구가 급격하게 증가하게 되었고 이에 따라 인터넷 사용량 또한 폭발적으로 증가하고 있는 추세이다. 이와 같이 증가하는 사용자를 수용하기 위해서는 네트워크 대역폭과 함께 서버의 용량을 충분하게 확보해야 한다. 그렇지 않을 경우 서버의 과부하 또는 네트워크의 병목 현상으로 인한 심각한 응답 지연으로 서비스의 만족도는 떨어지게 되며 상업적인 목적을 위해 인터넷 서비스를 제공하는 업체에게 이는 곧 금전적인 손실로 직결된다. 그러나 네트워크 대역폭과 서버의 용량을 충분히 확보한다고 하더라도 웹 트래픽 자체의 돌발적인 속성으로 인해 순간적으로 가해지는 부하의 변화가 극심하여 모든 상황에 완벽하게 대응할 수 있는 서버를 마련하는 것은 현실적으로 불가능하고 비용 또한 문제가 된다. 따라서 필연적으로 발생하는 서버의 과부하 상태를 극복하기 위한 방법이 필요하게 된다. 그러나 일단 과부하 상태에 진입한 서버는 큐에 있는 요청들을 제거하는 것만으로도 많은 부하를 발생시키게 되므로 이를 미연에 방지하기 위한 방안이 필요하게 된다. 따라서 일시적인 사용자 요청의 증가로 인해 과부하가 예상되는 시점에서 서버의 처리 용량을 넘어서는 트래픽에 대해서는 제한을 가하여 서버의 과부하 상태를 미연에 방지하고 부하가 최고치에 이르렀을 때에도 서비스 요청에 대해서 사용자가 만족할 수 있는 최소한의 응답 지연으로 서비스를 계속하고 최대한의 QoS(Quality of Service)를 보장하는 것이 필요하게 된다.

이러한 서버 과부하 문제를 해결하기 위해 기존의 연구는 크게 두 가지 흐름을 가지고 진행되어 왔다. 서버의 성능 향상을 통해 서버 과부하 문제를 해결하려는 방식이 그중 하나이다. 서버의 성능 향상을 위해 캐시 기법을 이용하고[1], 로드 밸런싱 기법을 이용하여 부하를 백 엔드 리얼 서버에 분산시키는[2, 3] 등의 기법이 연구되었다. 하지만 서버의 성능 향상을 통해 서버의 처리 용량을 확장하는 것이 가능하지만 근본적으로 서버 과부하의 문제가 해결된 것은 아니다. 따라서 서버에 유입되는 트래픽의 양을 제한하여 그로 인해 서버 내에 발생하는 부하를 적정한 수준으로 조절하여 보다 적극적으로 서버 과부하에 대응하는 방식에 대한 연구가 이루어졌다. 이러한 트래픽 제어 기법으로 트래픽 셰이핑[4, 5], 수락 제어 기법[6, 7, 8, 9] 등이 존재한다.

수락 제어 기법은 유입되는 트래픽이 증가하여 정해진 수준에 이르게 되면 클라이언트에 의한 서버로의 연결 요청 자체를 거부함에 의해 서버로 들어오는 트래픽을 제한하고 이를 통해 서버의 가해지는 부하를 조절하는 방식이다.

한편 트래픽 셰이핑 기법은 서버로 들어오는 트래픽의 유입률을 조절하는 방식으로 보통 패킷을 IP 주소에 따라 특정 클라이언트군과 연관시킨다. 각 클라이언트군은 서로 다른 우선순위를 가지며 우선 순위에 따라 해당 클라이언트군에 대한 패킷 유입률을 차등적으로 제한한다. 따라서 등급에 따라 차별적인 서비스를 제공하고 동시에 서버의 부하를 적정한 수준으로 유지하는 것이 가능하다.

본 논문에서는 일반적으로 서버의 과부하를 방지하기 위해 기존에 제시된 기법들에 대해 알아보고 그들의 장점과 단점을 고찰한다. 나아가 기존의 서버 과부하 방지를 위해 제시된 기법 중 수락 제어 기법을 이용하여 서비스 특성에 따라 다른 수락 제어 정책을 적용하고 이의 적용을 통해 QoS 측면에서의 향상 또는 throughput의 향상을 얻을 수 있는 방법을 소개한다.

서버의 부하 평가를 통해서 클라이언트의 연결 요청에 대해 수락 여부를 결정하는 방식은 기존의 수락 제어 기법과 동일하지만 서버 부하의 평가 시 기존의 수락 제어 기법이 클라이언트의 연결 요청 당시 현재의 서버 부하 상태에만 의존하여 수락 여부에 대한 결정을 내리는

것과는 달리 본 논문에서 제시하는 수락 제어 기법은 현재의 서버 부하 상태뿐만 아니라 그 이전의 서버 부하의 이력을 고려하여 클라이언트의 새로운 연결 요청에 대한 수락 여부를 결정한다. 이전의 서버의 부하 상태에 대한 고려를 수락 여부 결정 시 어느 정도 반영하는가에 따라 서버는 수락 제어 정책을 수정하는 것이 가능하고 트래픽 특성에 따라 다른 수락 제어 정책을 적용하는 것이 가능하게 된다. 또한 이전의 서버 부하의 이력에 대한 고려 정도에 따라 수락 제어 정책이 결정되고 이를 통해 서버는 QoS 측면에서의 향상을 얻거나 throughput의 증가를 이룰 수 있다. 이들은 서로 상보적인 관계를 가지고 있어 QoS의 향상 또는 throughput의 감소를 가져오며 반면 throughput의 증가는 QoS의 감소를 가져온다는 것을 보인다.

2. 제안 알고리즘의 설계 및 구현

2.1 기본 시스템 구성

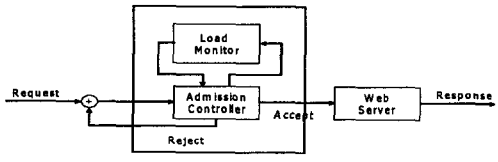

그림 1 기본 시스템 구성
Fig. 1 A basic system configuration

본 논문에서 제안하는 수락 제어 기법이 적용된 서버 시스템의 구조가 그림 1에 나타나 있다. 그림 12에서 보는 것과 같이 시스템은 부하 측정 장치와 수락 제어 장치로 구성된다. 클라이언트에 의해 연결 요청이 이루어지면 수락 제어 장치는 부하 측정 장치에 서버의 부하 상태에 대한 정보를 요청한다. 부하 측정 장치는 주기적으로 실행되면서 서버 시스템의 부하 상태에 대한 정보를 유지하고 있으면서 수락 제어 장치에 의해 요청을 받으면 서버의 부하에 대한 정보를 제공한다. 이를 바탕으로 수락 제어 장치는 클라이언트의 연결 요청에 대한 수락 여부를 결정한다.

2.2 수락 제어 장치 및 부하 측정 장치의 구조와 원리

- 서버와 클라이언트 간 연결 설정 과정

우선 서버와 클라이언트가 연결을 맺는 과정을 살펴보자. 먼저 ①서버로부터 서비스를 받기를 원하는 클라이언트가 서버에 TCP 연결을 요청하는 패킷(TCP SYN packet)을 전달한다. 패킷이 서버의 TCP에 전달된다. 서버의 TCP는 이에 응답하는 패킷(TCP SYN-ACK packet)을 클라이언트에 전송하고 새로운 socket을 생성한 후 listen queue의 SYN-RCVD queue에 새로이 생성한 socket을 놓는다. ②서버의 응답 패킷(TCP SYN-ACK packet)에 대해 클라이언트가 응답하는 패킷(TCP ACK packet)을 서버에 전송한다. 그러면 서버의 TCP는 해당 socket을 SYN-RCVD queue에서 제거하여 accept queue에 연결한다. ③웹 서버가 accept 시스템 콜을 호출할 때마다 listen queue의 accept queue에 있는 첫 번째 socket이 제거되어 웹 서버에 반환되고 서버와 클라이언트 간 연결이 설정된다[17, 18, 20].

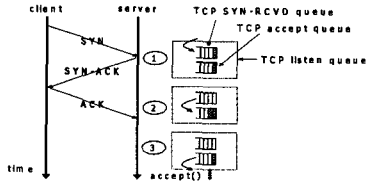

그림 2 클라이언트와 서버 간 연결 설정
Fig. 2 A connection setup between the client and the server

• 수락 제어 장치의 구조와 원리

위의 그림 2에서 설명한 것과 같이 클라이언트와 서버의 연결은 웹 서버가 accept 시스템 콜을 호출할 때 이루어진다. 따라서 수락 제어 장치의 구현은 accept 시스템 콜의 호출에 의해 서버가 클라이언트의 연결을 받아들이기 이전에 서버의 부하 상태에 대한 정보를 바탕으로 해서 해당 연결 요청을 수락하거나 거부하도록 해야 한다.

그림 3은 이를 바탕으로 하여 제안된 수락 제어 기법이 적용된 서버 시스템의 네트워크 계층 모델을 나타내고 있다. 그림 3에서 보는 것과 같이 수락 제어 장치는 리눅스를 운영체제로 하는 서버 시스템에 커널 모듈로써 포함되며 소켓 계층(socket layer)과 TCP/IP 스택의 사이에 위치한다. 사용자 영역의 응용 프로그램에게 소켓 인터페이스를 제공하는 소켓 계층의 일부 함수들과 네트워크 인터페이스 카드와 같은 물리 계층으로부터의 데이터 패킷을 가공하여 소켓 계층으로 전달하는 TCP/IP 스택의 일부 함수들을 사용하며, 두 계층 사이에 위치하게 된다.

위와 같이 커널 모듈로 삽입된 수락 제어 장치는 다음과 같은 과정을 통해 클라이언트의 연결 요청을 처리한다. 클라이언트가 서버에 연결을 요청하는 데이터 패킷을 전송한다. 서버는 클라이언트로부터 새로운 연결 요청을 받으면 서버의 부하 측정기가 부하를 평가한다. 평가된 부하가 미리 정해진 부하의 한계값 이상일 경우 해당 클라이언트의 연결 요청은 거부되고 서버의 TCP listen queue에서 제거된다. 이 경우 서버의 TCP는 과부하 상태를 알리는 분주 페이지로 클라이언트에 응답한다. 평가된 부하가 미리 정해진 부하의 한계값 이하일 경우 해당 클라이언트의 연결 요청은 수락되고 마찬가지로 서버의 TCP listen queue에서 제거되어 HTTP listen queue에 놓여진다. 서버와 클라이언트 사이에 연결이 설정되어 웹 서버는 해당 연결을 통한 서비스 요청을 받아들여 처리하고 클라이언트에 응답을 보낸다.

수락 제어 장치의 구현 원리는 다음과 같다. 수락 제어 기법을 사용하지 않을 경우 사용자 영역의 웹 서버와 소프트웨어와 TCP 스택은 다음의 그림 4와 같이 연결되어 진다.

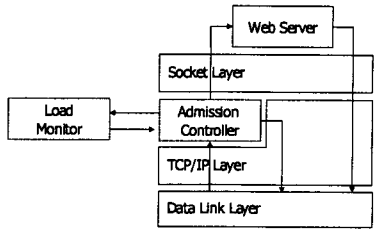

그림 3 수락 제어 기법의 계층 모델

Fig. 3 A layer model of the admission control scheme

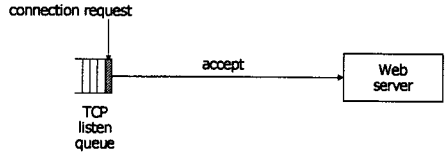

그림 4 일반 웹 서버의 구조

Fig. 4 An architecture of the web server

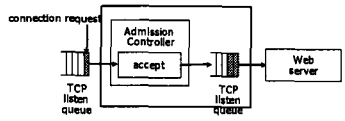

그림5 수락 제어 장치의 구조
Fig. 5 An architecture of the admission controller

수락 제어 기법을 구현하기 위해서 그림 5와 같이 사용자 영역의 웹 서버와 TCP 스택 사이에 TCP 스택과 웹 서버를 연결하는 부가적인 수락 제어 장치를 삽입한다. 삽입된 수락 제어 장치는 투명하게 구현되어 웹 서버나 TCP 스택은 수락 제어 장치가 삽입되기 전과 다름없이 동일하게 동작한다. 하지만 수락 제어 장치의 삽입으로 인한 약간의 부가적인 오버헤드의 발생을 감수해야 한다.

- 부하 측정 장치의 구현

부하 측정 장치는 수락 제어 장치와 마찬가지로 커널 모듈로 구현되어 매 초 실행되며 CPU 사용률에 대한 정보를 커널 내부 변수에 저장하고 갱신한다. 수락 제어 장치는 부하 측정기에 의해서 갱신되는 CPU 사용률에 대한 정보를 바탕으로 클라이언트의 연결 요청에 대해 수락 여부를 결정한다.

리눅스 운영체제는 시스템 내부적으로 CPU 사용률에 대한 정보를 유지하고 있다[15, 16]. 부하 측정기는 리눅스 운영체제 내에서 유지하고 있는 시스템 정보로부터 CPU 사용률에 대한 정보를 얻을 수 있다. 표 1은 시스템 정보를 유지하고 있는 커널 내부 변수와 그 구조체를 나타내고 있다. 표 2는 시스템 정보를 유지하는 커널 내부 변수를 이용하여 부하 측정기를 구현한 코드이다.

표 7 시스템 정보를 나타내는 커널 변수와 구조체
Table 1 Kenel variable and structure having the system information

```
struct kernel_stat {
   unsigned int per_cpu_user[NR_CPUS],
                per_cpu_nice[NR_CPUS],
                per_cpu_system[NR_CPUS];
   unsigned int
  dk_drive[DK_MAX_MAJOR][DK_MAX_DISK];
   unsigned int
  dk_drive_rio[DK_MAX_MAJOR][DK_MAX_DISK];
   unsigned int

dk_drive_wio[DK_MAX_MAJOR][DK_MAX_DISK];
   unsigned int

dk_drive_rblk[DK_MAX_MAJOR][DK_MAX_DISK];
   unsigned int

dk_drive_wblk[DK_MAX_MAJOR][DK_MAX_DISK];
   unsigned int pgpgin, pgpgout;
   unsigned int pswpin, pswpout;
#if !defined(CONFIG_ARCH_S390)
   unsigned int irqs[NR_CPUS][NR_IRQS];
#endif
   unsigned int ipackets, opackets;
   unsigned int ierrors, oerrors;
   unsigned int collisions;
   unsigned int context_swtch;
};
extern struct kernel_stat kstat;
```

표 8 부하 측정기를 구현한 코드
Table 2 A source code implementing the load monitor

```
static int ServerOverload(int thres)
{
    unsigned long jif = jiffies;
    unsigned int sum, use, sys, idl, tmp;
    static unsigned long comp = 95.;

    use = kstat.per_cpu_user[0]
kstat.per_cpu_nice[0];
    sys = kstat.per_cpu_system[0];
    idl = (jif - (use + sys))%UINT_MAX;
    sum = (use + sys + idl);
    tmp = (100*use)/sum + (100*sys)/sum;

    comp = (1-0.1)*comp + 0.1*tmp;

    if ( comp > thres)
      return 1;
    else
      return 0;
```

3. 검증 및 분석

3.1 기본 실험 환경

이 장에서는 제안된 수락 제어 알고리즘의 타당성 여부를 검증하기 위해 그림 6과 같은 실험 환경을 구성하였다. 세 대의 클라이언트는 트래픽을 발생시키고 정해진 규칙에 따라 웹 서버에 데이터를 요청한다. 이때 요청되는 데이터의 양은 웹 서버의 처리 용량을 초과할 수 있을 정도까지 충분하게 점진적으로 증가시켜 웹 서버를 과부하 상태에 이를 수 있도록 한다. 웹 서버가 클라이언트의 요청을 어느 정도 처리할 수 있는 지의 여부를 측정하기 위해 벤치마킹 프로그램을 사용하여 측정한다.

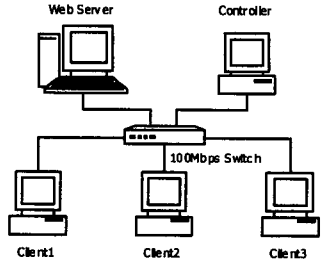

그림 6 테스트 환경
Fig. 6 Test environments

벤치마크 프로그램으로는 유닉스 환경에서 사용이 가능한 httperf 0.8과 autobench를 사용한다[13, 14]. 실험을 총괄하기 위해 한 대의 컨트롤러를 필요로 한다. 1대의 컴퓨터를 컨트롤러로 설정하여 실험을 전체적으로 제어하고 결과 데이터를 수집하도록 한다.

실험에서 네트워크가 포화되어 병목현상이 발생하는 것을 방지하기 위해서 3com SuperStack 3 Switch를 사용하여 100Mbps 기반의 고속 이더넷 환경을 구축하여 충분한 네트워크 대역폭을 확보한다. 이를 통해 실험에서 네트워크에 대한 고려를 배제하는 것이 가능하다. 웹 서버에는 낮은 사양의 Celeron 300MHz 프로세서와 256Mbyte 메모리를 사용하였다.

세 대의 클라이언트가 웹 서버에 트래픽을 발생시킬 수 있는 환경을 마련하였고 메모리를 충분하게 장착하였다. 웹 서버 응용 프로그램으로는 Apache 1.3.9[19]를 운영체제는 Linux 7.1(kernel 2.4)[22]을 설치하였고 웹 서비스를 제외한 다른 서비스를 실행되지 않도록 하였다. 세 대의 클라이언트는 Pentium III 1GHz 프로세서와 256Mbyte 메모리, 3Com 3CSOHO100-TX NIC을 사용하였고 한 대의 컨트롤러는 Athlon 700 MHz 프로세서와 256Mbyte 메모리, 3Com 3CSOHO100-TX NIC을 사용하였다. 클라이언트의 경우 서버에

트래픽을 발생시키는 것은 내부적으로 큰 부하를 발생시키므로 높은 사양의 프로세서가 필요하지만 컨트롤러는 실험을 제어하고 결과 데이터를 수집하는 일을 하며 이는 높은 사양의 컴퓨터를 필요로 하지 않는다. 실험을 위해 준비된 환경을 정리하면 표 3과 같다.
α의 값이 0.1, 0.5, 1.0인 경우에 대해 실험을 진행하고 수락 제어 알고리즘을 위한 파라미터로 다음의 값을 사용한다.

- LThreshold = 95 (%)
- T1 = T2 = ... = Ti = ... = 1 (초)

제안된 수락 제어 기법이 적용된 서버에 부하를 단계적으로 증가시키면서 throughput과 QoS를 측정하고 α의 값의 변화에 따라 나타나는 결과를 비교 분석한다.

그림 7은 α 값의 변화에 따라 제안된 수락 제어 기법이 적용된 서버의 QoS를 측정한 결과이다.

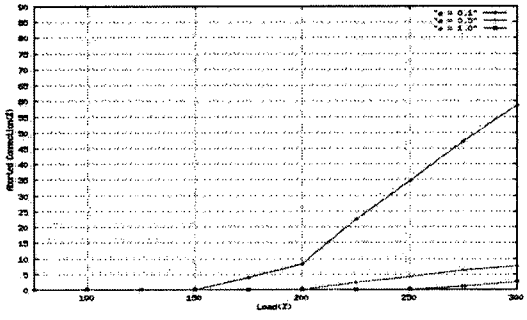

그림 7 α 값의 변화에 따른 중단된 연결의 비율

Fig. 7 A percentage of aborted connections while varying α

표 9 테스트 조건

Table 3 Test conditions

종류	구분	내용
서버	운영체제	Redhat Linux 7.1
	프로세서	Intel Celeron 300MHz
	메모리	256MB
	네트워크	Realteck RTL8139A 100Mbps NIC
컨트롤러	운영체제	Redhat Linux 7.1
	프로세서	AMD Athlon 700MHz
	메모리	256MB
	네트워크	3Com 3CSOHO100-TX 100Mbps NIC
클라이언트	운영체제	Redhat Linux 7.1
	프로세서	Intel Pentium-III 1GHz
	메모리	256MB
	네트워크	3Com 3CSOHO100-TX 100Mbps NIC
스위치	스위치	3Com SuperStack 3 100Mbps Switch
벤치마크	벤치마크	httperf 0.8
		http://www.hpl.hp.com/personal/David_Mosberger/httperf.html

3.2 정책에 따른 수락 제어 알고리즘의 QoS 비교

수락 제어 기법에서 QoS는 특정한 사용자 또는 클라이언트에 대해서 대역폭을 할당하고 이를 보장하는 일반적인 의미의 QoS를 의미하지 않는다. 수락 제어 기법에서는 사용자 또는 클

라이언트를 식별할 수 있는 방법이 없다. 따라서 수락 제어 기법에서 의미하는 QoS는 연결이 허용된 사용자 또는 클라이언트에 한하여 그 사용자 또는 클라이언트가 성공적으로 서비스를 종료할 때까지 서버의 부하를 적정한 수준으로 유지하여 서비스를 받는 도중 연결이 끊어지지 않도록 최대한 보장하는 것을 의미한다. 따라서 수락 제어 기법에서는 클라이언트가 서버로부터 서비스를 받는 도중 클라이언트와 서버 사이의 연결이 끊어지지 않는 것을 보장하지 않는다. 다만 최대로 보장할 뿐이다. 따라서 수락 제어 기법이 적용된 서버의 QoS를 측정하기 위해 서비스를 받는 도중 끊어지는 연결의 비율을 측정하여 QoS 보장 정도를 측정하였다.

그림 18의 결과는 서버의 부하가 증가함에 따라 중단된 연결의 비율이 증가함을 보여주고 있다. 즉 서버의 부하가 증가함에 따라 서버 내에서 처리해야 하는 요청의 양이 증가로 중단된 연결의 비율이 증가하는 것을 나타내고 있다. 서버 부하가 증가함에 따라 QoS 가 감소하는 것을 나타내는 것이다.

α 값의 변화에 따르는 QoS의 변화를 살펴보면 α의 값이 증가함에 따라 중단된 연결의 비율이 감소하여 더 나은 QoS 수준을 나타내고 있다.

3.3 정책에 따른 수락 제어 알고리즘의 throughput 비교

그림 8은 α 값의 변화에 따라 제안된 수락 제어 기법이 적용된 서버의 throughput을 측정한 결과이다. throughput은 수락된 전체 연결 중에서 성공적으로 서비스를 종료한 연결의 비율로서 측정되었다. 그림 19의 왼쪽 부분 즉 부하의 수준이 낮을 때에는 α 값의 증가에 따라 throughput이 감소하고 있고 오른쪽 부분 즉 부하의 수준이 높을 때에는 α 값의 증가에 따라 throughput이 증가하고 있는 모습을 보여주고 있다.

그림 19의 부하의 수준이 낮은 영역을 보면 α 값이 작을 경우 수락 여부 결정은 이전의 서버 부하 상태를 더 많이 고려하므로 부하 변동에 따라 수락 제어 정책의 변화가 느리다. 따라서 α 값이 큰 경우보다 수락되는 연결은 많아지고 거부되는 연결은 더 적어진다. 따라서 throughput은 증가하게 된다. 하지만 부하의 수준이 증가하게 되면 이러한 결과를 반대로 나타난다. 이것은 그림 19에서 나타난 바와 같이 중단된 연결이 증가하기 때문이다. 중단된 연결의 증가로 인해 비율로서 측정되는 throughput은 감소하게 된다.

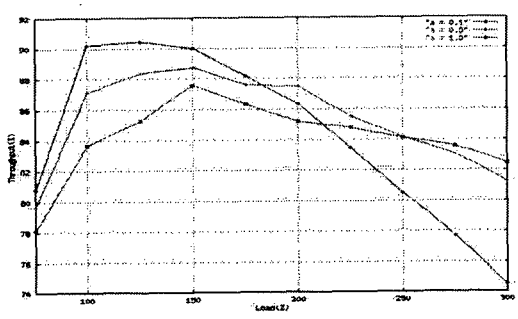

그림 8 α 값의 변화에 따른 완료된 연결의 비율

Fig. 3 A percentage of completed connections while varying α

4. 결 론

수락 제어 알고리즘은 웹 서버의 QoS를 보장하기 위해서 사용되는 기법이다. 수락 제어 기법을 사용하여 서버의 부하를 제한 또는 조절할 경우 너무나 당연하게 웹 서버의 QoS를 향상시키는 것이 가능하다. 하지만 어느 정도의 throughput의 희생은 감수되어야 한다.

본 논문에서는 정책 기반의 수락 제어 알고리즘을 제안하였다. 제안된 수락 제어 알고리즘을 통해서 말하고자 하는 것은 웹 서버 QoS를 보장하는 최적의 수락 제어 알고리즘을 찾고자 하는 것이 아니다. 서버의 부하 평가에 있어서 어려움과 서버 처리 용량의 설정이 명확하게

이루어지기 어려움으로 인해 일반적으로 통용될 수 있도록 하는 QoS 보장 수락 제어 알고리즘을 찾는 것은 매우 어려운 일이고 경우에 따라 다를 수 있어 시스템 자원을 가장 효율적으로 이용하는 최적의 알고리즘을 찾는 것은 매우 어렵다. 따라서 본 논문에서는 트래픽이나 또는 서비스 요구 조건에 따라 다른 수락 제어 정책을 적용할 수 있는 정책 기반의 수락 제어 알고리즘을 제안하였다. 제안된 수락 제어 알고리즘의 α의 값을 조율함에 따라 서버 부하의 평가 시에 현재의 부하 상태만을 고려할 것인가 이전의 부하 상태를 함께 고려할 것인가를 결정할 수 있고, 그에 따라 클라이언트의 서비스 요청에 제한적으로 반응하는 수락 제어 정책을 선택할 수도 있고 또는 클라이언트의 서비스 요청에 허용적으로 반응하는 수락 제어 정책을 선택할 수도 있다. 본 논문에서는 이를 각각 시스템 제한성 강조 수락 제어 정책, 시스템 허용성 강조 수락 제어 정책으로 칭하였다. 또한 각 정책은 정책에 따라 QoS를 향상시키는 것이 가능하거나 throughput을 향상시키는 것이 가능하기도 하다. 서비스 요청에 제한적으로 동작하는 정책을 사용할 경우 거부되는 연결의 수는 증가하고 수락되는 연결의 수는 감소하지만 중단되는 연결의 수는 감소하므로 throughput의 향상이 있지만 QoS 측면에서의 희생이 필요하게 된다. 서비스 요청에 허용적으로 동작하는 정책을 사용할 경우는 반대로 거부되는 연결의 수는 감소하고 더 많은 연결을 수락하며 중단되는 연결의 수는 그만큼 더 늘어나므로 QoS 측면에서의 향상이 있지만 throughput은 감소하게 된다.

따라서 QoS의 보장을 더 강조하는가 또는 throughput의 향상을 더 중요시하는가에 따라 서로 다른 수락 제어 정책을 적용할 수 있게 된다. 수락 제어 함수는 이러한 수락 제어 정책을 결정할 수 있게 한다. α의 값에 따라서 수락 제어 함수는 현재의 서버 부하 상태에 의존하여 수락 여부를 결정할 수도 있고 현재의 서버 부하 상태뿐만 아니라 이전의 서버 부하 상태 즉 서버의 부하 이력을 함께 고려하여 수락 여부를 결정할 수도 있다. 즉 α의 값이 큰 경우 서버는 현재의 부하 상태에 따라 수락 여부를 결정하게 되고 따라서 서버는 부하 상태의 변화에 따라 빠르게 반응하게 되며 α의 값이 작은 경우 서버는 현재의 부하 상태뿐만 아니라 이전의 서버 부하 상태를 함께 고려하므로 서버의 부하 상태의 변화에 빠르게 반응하는 것이 어렵게 된다.

또한 큰 α값을 가지는 수락 제어 기법의 경우 상대적으로 높은 수준의 QoS를 얻을 수 있는 반면 throughput의 감소를 감수해야 하고 작은 α값을 가지는 수락 제어 기법의 경우 상대적으로 throughput의 향상이 있을 수 있지만 더 낮은 수준의 QoS를 감수해야 한다. 결국 제한성이 강조되는 서비스인지 아니면 허용성이 강조되는 서비스인지에 따라 다른 수락 제어 정책을 적용하는 것이 가능하다. 즉 다시 말하면 제안된 수락 제어 알고리즘에서는 α의 값을 조절하면 각 서비스의 특성에 따라 서로 다른 정책을 가진 수락 제어 정책을 사용할 수 있게 된다. 하지만 어떠한 정책도 완전하지는 않다. QoS의 향상을 얻기 위해서는 throughput의 감소를, throughput을 증가시키기 위해서는 QoS 측면의 손실을 감수해야 한다. 이렇게 QoS와 throughput은 서로 tradeoff 관계에 있게 된다.

참고 문헌

[1] G. Barish, K. Obraczke, "World Wide Web caching: trends and techniques," IEEE Communications Magazine, Volume. 38, Issue. 5, pp. 178-184, May 2000.

[2] S. Ralf, "Load Balancing Your Web Site Practical Approaches for Distributing HTTP Traffic", http://www.webtechniques.com/archives/archives/1998/05/engelschall/

[3] Tony Bourke, "Server Load Balancing", O'Reilly & Associates, Inc., 2001.

[4] Hani J, John R, Kang G. Shin, "QGuard: Protecting Internet Servers from Overload" Realtime Computing Laboratory, University of Michigan, 2000.
[5] V. Fineberg, "A practical architecture for implementing end-to-end QoS in an IP network", IEEE Communications Magazine, Volume. 40, Issue. 5, pp. 122-130, Jan. 2002.
[6] L. Ckerkasova, P. Phaal, "Session Based Admission Control: a Mechanism for Improving Performance of Commercial Web Sites", Technical Report, Hewlett Packard, 1999.
[7] N. Bhatti and R. Friedrich, "Web server support for tiered services", IEEE Network, Volume. 13, Issue. 5, pp. 64-71, Sept. 1999.
[8] T. Voigt, P. Gunningberg, "Kernel-based Control of Persistent Web Server Connections", ACM Performance Evaluation Review, pp. 20-25, Sept. 2001.
[9] T. Voigt, R. Tewari, D. Freimuth, A. Mehra, "Kernel Mechanisms for Service Differentiation in Overloaded Web Servers", 2001 Usenix Annual Technical Conference, Boston, MA, USA, June, 2001.
[10] William Stalling, "HIGH-SPEED NETWORKS: TCP/IP and ATM Design Principles", Prentice-Hall International, Inc., 1998.
[11] http://kldp.org/Translations/html/Netfilter-hacking-KLDP
[12] http://kldp.org/Translations/IPCHAINS-HOWTO
[13] httperf , http://www.hpl.hp.com/personal/David_Mosberger/httperf.html
[14] autobench, http://www.xenoclast.org/autobench
[15] Daniel P. Bovet, Marco Cesati, "Understading the LINUX KERNEL", O'Reilly & Associates, Inc., 2000.
[16] Gary Nutt, "Kernel Projects FOR Linux", Addison Wesley Longman, Inc.
[17] Behrouz A. Forouzan, "TCP/IP Protocol Suite", McGraw-Hill Companies, Inc.
[18] kHTTPd, http://www.fenrus.demon.nl.
[19] Apache, http://www.apache.org
[20] Gaurav Banga, Peter Druschel, "Measuring the Capacity of a Web Server", In Proceedings of USITS, Dec., 1997.
[21] 이철훈, 이일신, "웹서버의 고가용성 보장 기술", 전기학회지 Vol. 49, No. 10, pp 4-8, 2000.
[22] Linux Virtual Server Project, http://www.linux-vs.org
[22] Redhat Linux, http://www.redhat.com