SLA를 고려한 웹 서버 성능 분할 기법

고현주 °*, 박기진** , 박미선*

안양대학교 문리과학대학 컴퓨터학과*
아주대학교 공과대학 산업정보시스템공학부**
e-mail : khj02@anyang.ac.kr°*, kiejin@ajou.ac.kr**, oooooo82@anyang.ac.kr*

Performance Isolation Technique Considering Service Level Agreement in Web Servers

Hyeonjoo Go°*, Kiejin Park**, Misun Park*

Dept. of Computer Engineering, Anyang University*
Division of Industrial & Information Systems Engineering, Ajou University**

요약

클라이언트와 서비스 제공자간의 서비스 수준 계약인 SLA(Service Level Agreement)를 만족시키기 위해서는 클라이언트 요청을 우선순위 클래스로 구분하여, 낮은 수준의 서비스를 요청하는 클라이언트 보다는 고수준의 서비스를 요구하는 클라이언트에게 우선적으로 서비스를 제공할 수 있는 기술이 필요하다. 본 논문에서는 서비스 제공자의 웹 서버 노드를 우선 순위에 따라 정적·동적으로 분할하는 방법을 연구하였으며, 시뮬레이션을 통해 SLA를 고려할 수 있는 응답시간 성능을 분석하였다.

1. 서론

웹 서버에서 인터넷 비즈니스와 관련되어 복잡하고 중요한 서비스를 제공하기 위해서는 신뢰성 있는 고품질의 서비스 수준(Service Level)을 고려할 수 있어야 한다. 현재까지 고품질의 서비스 수준을 제공하기 위한 연구는 대부분 네트워크 단계에서 주로 다루어졌으나, 웹 서버에서는 클라이언트 요청을 우선 순위 방식이 아닌 선입 선출(FIFO) 방식으로 처리하기 때문에 시스템의 과부하 상황에서는 우선 순위가 높은 클라이언트 요청이 거절되는 경우가 발생한다. 따라서, 네트워크 단계의 QoS 만으로는 양단간 QoS 를 지원할 수 없는 문제점을 해결하기 위하여 웹 서버 단계에서 클라이언트 요청의 우선 순위에 따라 차별화된 서비스를 제공할 수 있는 기법이 필요하다.

웹 상에서 차별화된 서비스(Differentiated Service)를 제공하기 위하여 웹 서버 단계의 QoWS(Quality of Web Services)에 관한 분석이 또한 필요하며[1], QoWS 는 사용자 또는 어플리케이션에 대해 중요도에 따라 서비스 수준을 차등화하여 한정된 WAN 대역폭에서 트래픽과 대역폭을 정책적으로 관리하는 기술이다. SLA(Service Level Agreement)는 서비스 제공자와 대상 고객 간 계약으로, 제공되어질 서비스와 그와 연관된 조건들을 사전에 약속하여 그 만큼의 서비스를 제공하는 것이며, 차별화된 서비스를 통해 고객과 서비스 제공자간의 SLA 보장이 가능하다. 클러스터링된 여러대의 서버 노드로 구성된 웹 서버에서는 클라이언트의 요청 수준에 따라 그에 알맞은 특정 서버 노드들을 할당함으로써, 효율적인 차별화 서비스를 구현할 수 있다.

본 연구는 한국학술진흥재단 신진교수연구과제(KRF-2003-003-D00331)지원으로 수행되었음.

이러한 서버 분할 기법은 QoWS 를 보장해주는 중요한 요소중 하나인 성능 분리(Performance Isolation)에 해당하며, 분할된 서버마다 서로 다른 클래스의 서비스를 담당하게 하고, 상위 계층의 클라이언트 요청일수록 더 많은 서버를 할당해 줌으로써, 다중클래스(Multiclass)별로 SLA 를 보장할 수 있게 된다.

본 논문의 2 장에서는 관련 연구를 언급하고, 3 장에서 웹 서버의 구조에 대해 알아보고, 4 장에서는 웹 서버 분할 기법을 제시 하였으며, 5 장에서는 제안한 방법의 성능 평가를 수행하고, 6 장에는 결론을 내렸다.

2. 관련 연구

클러스터 웹 서버 구조에서 우선 순위별로 클라이언트 요청을 처리하기 위해, 서버 노드를 정적 · 동적으로 분할하는 기법이 연구되고 있다. 전자는 클라이언트의 계층별 요청 수준에 따라 고정된 수의 서버 노드를 클래스별로 할당하는 방법으로 클라이언트의 계층별 요청 수가 자주 변하는 웹 환경에는 적당하지 않다[2]. 이는 특정 클래스에 할당된 서버 노드가 과부하 상태인 반면에 다른 클래스의 서버 노드는 유휴 상태에 있을 수 있기 때문이다.

동적 분할 기법은 클라이언트의 계층별 요청 수준 혹은 필요에 따라서 동적으로 서버 노드를 할당함으로써, 높은 자원 이용률을 제공한다. 동적 분할 기법에는 DDSD(Demand-driven Service Differentiation)[3], Dynamic Partitioning[4] 기법등이 있으며, DDSD 기법은 클라이언트 요청량에 따라 CPU 와 디스크 I/O 용량을 할당함으로써 차별화된 서비스를 제공하고, e-비즈니스와 관련된 웹 사이트 컨텐츠처럼 CPU 와 디스크 I/O 의 처리를 많이 요구하는 동적 요구가 많은 상황에 적합하다. Dynamic Partition

ing 기법은 SLA를 만족하는 서비스를 상위 클래스 사용자에게 제공하기 위해 서버의 부하량에 따라 서버 노드를 동적으로 분할하는 기법이다. 하지만 이들은 정적 요구를 전혀 고려하지 않고 동적 요구만 고려했기 때문에 정적 요구가 많아질 경우 성능 저하가 발생하는 문제점이 있다. 대체적으로 동적 분할 기법은 정적 분할 기법 보다 높은 자원 이용률을 제공하지만, 정적 요구가 많아질 경우 정적 분할 기법이 더 좋은 성능을 제공한다.

클라이언트의 요청을 2 개의 계층(High set, Low set)으로 분리하여, 차별화된 서비스를 제공하려는 시도가 있었으며[4], 본 논문에서는 클라이언트 계층을 단순히 2 개로만 구별하지 않고, 이를 일반화시킨 개념인 다중 클래스 문제를 다루었다. 즉, 서버 노드가 클라이언트의 계층별 우선 순위에 따라 분할되며, 클라이언트 계층별 요청률에 따라 분할된 서버 노드들을 정적·동적으로 관리하는 기법에 관하여 분석하였다.

3. 웹 서버의 구조

그림 1은 본 논문에서 고려하고 있는 웹 서버 클러스터의 연결 구조이다. 로드발런서는 서비스 노드(Real Server)들 간에 작업들을 고루 분배하는 기능을 수행하며, 과중하게 부하를 받고 있는 노드들이 다른 노드들에게 작업을 전달하거나, 유휴상태에 있는 노드의 작업분배 요청을 처리한다. 이러한 웹 서버 구성 방식은 웹 클라이언트로 하여금 하나의 컴퓨터 즉, 로드밸런서로부터 서비스를 제공받는 것으로 여겨지게 하지만, 실제 서비스 제공은 로드밸런서에 연결되어 있는 서비스 노드들에 의해서 이루어진다.

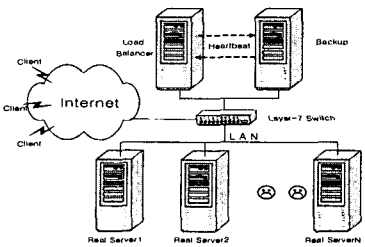

그림 1. 웹 서버 클러스터 구조

4. 웹 서버 분할

본 장에서는 n개의 우선순위 클래스로 분류된 클라이언트 요청을 처리하기 위해, 정적·동적으로 서버를 분할 하는 기법에 관한 내용을 기술하고자 한다.

4.1 서비스 분류(Service Classification)

서비스 분류는 로드발런서에 입력되는 클라이언트의 요청을 분류하여 계층별 큐(Queue)로 보내는 작업을 의미하며, 다양한 클라이언트 계층에 대한 차별화된 서비스를 제공하기 위해 다중클래스 개념을 도입하였다. 클라이언트 요청은 우선 순위에 따라 n 개의 클래스로 구분되며, 각각의 클래스를 담당할 서버에 할당된 클라이언트의 요청이 들어오면, 그 요청의 우선 순위를 확인한 후에 그에 적합한 서버에서 요청을 처리하며, 전체 서버 수가 N 대인, 서버 할당 방식을 수식으로 나타내면 식 (1)과 같다 여기서, $CS_i(t)$ 는 임의의 시간 t 에서 클래스 i 에 할당되는 서버의 수이며, $CS_{i-1}(t)$ 는 $CS_i(t)$ 보다 우선순위가 높다. 또한 $us_i(t)$ 는 임의의 시간 t 에서 클래스 i 에 속하는 클라이언트 요청을 처리하는 서버를 구분하는 경계값이다.

$$
\begin{aligned}
&CS_1(t) = \{1, \Lambda\ , us_1(t)\}, \\
&CS_2(t) = \{us_1(t)+1, \Lambda\ , us_2(t)\}, \\
&\qquad \Lambda \\
&CS_n(t) = \{us_{n-1}(t)+1, \Lambda\ , N\}, \\
&us_i(t) \in \{1, \Lambda\ , N\},\ us_1(t) \geq 1,\ us_{n-1}(t) \leq N-1,\ us_i(t) \leq us_{i+1}(t) \qquad (1)
\end{aligned}
$$

이와 같이 서버 노드들은 클라이언트 요청 우선 순위에 따라 n 개의 클래스로 구분함으로써, SLA를 만족하는 클라이언트의 계층별 서비스를 제공하고자 한다.

4.2 정적 분할(Static Partition)

정적 분할 기법은 과거의 계층별 클라이언트의 요청률을 고려하는 서버 할당 방식으로, 시스템의 최초 가동시에 각각의 클라이언트 계층별로 고정적으로 서버를 할당하며, 한번 할당된 서버 노드는 클라이언트의 계층별 요청률이 변해도 수정되지 않는다. 식 (2)는 정적 분할 방식을 나타내고 있으며, $us_i(0)$ 는 시스템 최초 가동시에 정해지는 값으로, 최초 웹서버를 가동시켰을 때 정해지는 i 번째 클래스를 서비스하는 서버 경계값이다(식 (1)참조). 여기서 ρ_i 는 전체 요청 수와 i 클래스에 속하는 클라이언트의 요청 수의 비율이고, T_N 은 N 대의 서버에서 허용 가능한 최대 지연 시간이다. 또한, $MaxConn(T_N)$ 은 지연 시간 T_N 을 만족시키는 최대 접속 수이며, T_i 는 i 번째 클래스의 SLA 를 고려한 임계 시간(Threshold Time: $T_i < T_N$)이다.

일반적인 상황에서는 항상 T_N 에서의 최대 접속수를 만족하지 않기 때문에, T_N 에서의 이상적인 최대 접속수를 T_i 에서의 실제 최대 접속수로 나누어, 최대 접속 비율을 구한다. 기본적으로 서버의 수를 구하기 위해서 ρ_i 와 전체 서버대 수를 곱하며, 최대 접속 비율을 곱하여 더 정확한 값을 끌어낸다.

$$
us_i(0) = \lceil \rho_i \cdot N \rceil \cdot \frac{MaxConn(T_N)}{MaxConn(T_i)},\ i = 1, \cdots, n-1 \qquad (2)
$$

식 (2)와 같은 정적 분할 방식은 특정 웹 서버 노드에 과부하가 걸렸다는 것을 웹 관리자가 인지하기 전까지 $us_i(0)$ 값을 계속 그대로 유지하는 문제점을 가지고 있다.

4.3 동적 분할(Dynamic Partition)

동적 분할 기법은 상위 계층의 사용자의 SLA를 만족하는 서비스를 제공하기 위해, 클래스별 부하량에 따라 서버 노드를 동적으로 분할하는 기법으로, 특정 클래스를 서비스하는 노드에 과부하가 걸렸을 경우, 이보다는 더 낮은 클래스를 서비스 하고 있는 서버를 추가적으로 이용하거나, 과부하 상황이 해소되면 다시 서버를 반납하는 것을 기본 개념으로 하고있다. 동적 분할 개념은 부하 변화에 맞게 대처할 수 있어 여러 사용 계층자들

에게 효율적이면서도 질 높은 서비스를 제공할 수 있다.

$SumLoad_{CS_i}(t)$는 $CS_i(t-1)$에서의 서버 부하(Load)의 합이며, 부하는 클라이언트의 최대 접속 수($MaxConn$)를 의미한다. 해당 클래스의 부하합이 작업 능력을 넘어서면, 과부하 상태로 판단하며 기존에 할당받은 서버로는 효율적으로 일을 처리할 수 없게 된다.

$$Diff = SumLoad_{CS_i}(t) - (us_i(t-1) - us_{i-1}(t-1)) \cdot MaxConn(T_i);$$
$$\text{If}(\,|\,Diff\,|\, > SumLoad_{CS_i}(t) \cdot \alpha)\{$$
$$\text{If}\,(SumLoad_{CS_i}(t) > (us_i(t-1) - us_{i-1}(t-1)) \cdot MaxConn(T_i))$$
$$us_i(t) = us_i(t-1) + 1;$$
$$\text{else if}(SumLoad_{CS_i}(t) < (us_i(t-1) - us_{i-1}(t-1)) \cdot MaxConn(T_i))$$
$$us_i(t) = us_i(t-1) - 1;$$
$$\} \qquad (3)$$

식 (3)은 계층 i를 서비스 하는 서버의 동적 분할 방식을 표시하고 있으며, 부하의 변동 비율(α)에 따라 클라이언트 계층별 서버 노드의 경계값이 계산된다. 즉, 실제 부하량과 처리 용량과의 차이를 구하여, α %이내의 변동은 서버 분할에 영향을 미치지 않도록 하였다. 이와 같이 동적으로 서버 노드를 분할하려면, 과부하 상태시 어떤 클래스에서 서버를 제공받을 것인지, 혹은 반대의 경우 누구에게 돌려줄 것 인지도 고려해야 하며, 본 논문에서는 특정 클래스의 클라이언트의 요청률이 높아져 과부하가 걸릴 경우, 과부하가 걸린 서버보다 한 단계 낮은 우선 순위의 클래스에서 서버를 가져가고, 클라이언트의 요청이 적어지면 다시 한 단계 낮은 우선 순위의 클래스로 반납하는 방식을 채택하였다(그림 2 참조).

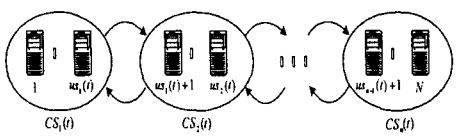

그림 2. 서버 노드의 획득과 반납

5. 성능 평가

실제적인 웹 서버 클러스터 시스템의 성능 분할을 원활히 시뮬레이션 하기 위해, 클라이언트의 도착률에 따른 응답시간 및 승인 거절된 요청 수에 관하여 분석하였으며, 95-percentile은 SLA를 고려하기 위해 사용된 척도로써, 들어온 요청중 95% 이상은 서비스 제공자와 고객간의 계약된 시간 안에 응답시간을 만족한다는 것을 의미한다. 웹 서버는 각 클라이언트 계층으로부터 정적 요청과 동적 요청을 받아들이며, 시뮬레이션에 사용된 웹 서버 클러스터 시스템의 운영 파라미터는 표 1, 2과 같다[2].

표 1. 정적·동적 요청 부하 모델

Type	Mean service time	Frequency
Dynamic requests	700 msec.	0.20
Static requests	100 msec.	0.80

표 2. 시스템 파라미터 (단위:second)

서비스 요청 도착 시간 간격	Exp(평균 서비스 시간)
서버 대수	10
클래스별 요청 도착 비율 $[class_H, class_M, class_L]$	[0.2, 0.3, 0.5]

그림 3에서는 3개의 클래스(High, Medium, Low)를 대상으로, High 클래스와 Medium 클래스, Low 클래스를 정적·동적 분할에 따른 응답 시간을 클라이언트의 도착률의 변동에 따라 표시하였다. 정적 분할보다는 동적 분할 기법의 응답시간이 모든 클래스에서 작게 나왔으며, 이는 동적 분할 기법이 사용자 요청을 고려하여 부하 변화에 맞게 대처하였기 때문이라 판단한다.

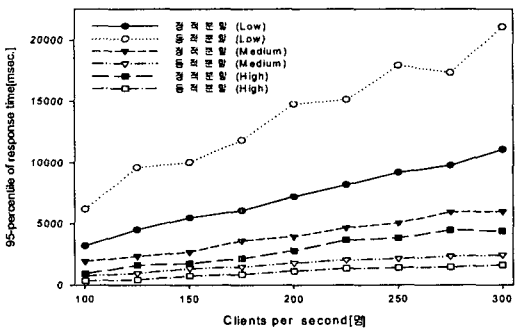

그림 3. 도착률에 따른 분할기법 간의 응답 시간 비교

6. 결론

특정 서버로의 과부하는 시스템 다운이나 심각한 응답 지체 현상과 같은 상황을 발생시키기 때문에, 이러한 문제를 피하기 위해서는 서버를 동적으로 분할해야하며, 클라이언트 계층을 일반화시켜야한다. 본 연구에서는 클라이언트 계층별 요청률에 따라 분할된 서버 노드들을 정적·동적으로 관리 함으로써, 웹 서비스의 응답 시간과 시스템 성능을 향상시켰으며, 추후에는 서비스 요청 시간대 따라 부하를 적절하게 조절할 수 있는 동적 분할 기법에 대한 연구를 수행할 예정이다.

참고 문헌

[1] V. Cardellini, E. Casalicchio, M. Colajanni, and M. Mambelli, "Enhancing a Web-server Cluster with Quality of Service Mechanisms," Proceedings of 21st IEEE International Performance, Computing, and Communications Conference, pp. 205-212, Apr. 2002.

[2] V. Cardellini, E. Casalicchio, M. Colajanni, and M. Mambelli. Web switch support for differentiated services. *ACM Performance Evaluation Review*, 29, 2001.

[3] H. Zhu, H. Tang, and T. Yang. Demand-driven service differentiation in cluster-based network servers. In *Proc. of IEEE Infocom 2001*, Anchorage, Alaska, Apr. 2001.

[4] M. Andreolini, E. Casalicchio, M. Colajanni and M. Mambelli, A Cluster-Based Web System Providing Differentiated and Guaranteed Services, Cluster Computing, 7(1): 7-19, Jan. 2004

[5] LVS Project, www.linuxvirtualserver.org