성능향상을 위한 CGI 애플리케이션 구조에 관한 연구

임인택*, 정영석*, 윤경희*, 김종근*
*영남대학교 컴퓨터공학과

A Study on CGI Application Structures for Performance Improving

Intaek Leem*, Youngseok Jung*, Kyunghee Yun*, Chonggun Kim*,
*Dept of Computer Engineering, Yeungnam University

요 약

동적 웹 컨텐트(dynamic web content)를 생성하는 CGI 애플리케이션은 사용시 웹 서버의 자원을 필요로 하기 때문에 서비스를 요청하는 클라이언트의 수가 많을수록 프로세스의 크기 및 프로세스 관리 부담이 가중되어 전체 시스템의 성능이 저하되는 단점이 있다. 본 논문에서는 CGI가 갖는 프로세스 생성 및 관리비용을 최소화하고, 캐싱을 이용하여 성능을 향상시킬 수 있는 CGI 애플리케이션 구조를 제안하였다. 그리고 제안한 CGI 방식의 성능 분석 결과, 제안한 방식이 기존의 CGI 방식보다 빠른 응답시간과 메모리를 효율적으로 사용하여 다중 사용자 환경에서 더 우수한 결과를 나타내었다.

1. 서 론

최근 웹의 폭발적인 사용은 동적 웹 컨텐트를 생성하는 CGI[1] 애플리케이션에 대한 많은 요구를 발생시켰다. 특히 웹을 통한 S/W 배포, 전자상거래 등이 일반화되고 있다. 정적 웹 컨텐트로부터의 이러한 이동은 여러가지 제한에 부닥치고 있으며, 현재 이들 애플리케이션의 성능을 결정하는 포인트가 CGI이다. CGI를 통해 실행되는 애플리케이션들은 많은 웹 서버의 자원을 필요로 하기 때문에 서비스를 요청하는 클라이언트의 수가 많을수록 프로세스의 크기 및 프로세스 관리 부담이 가중되어 전체 시스템의 성능이 저하되는 단점이 있다. 이러한 CGI의 단점을 개선하기 위한 여러 연구가 진행되고 있다[2][3]. 또한 CGI를 다양한 목적으로 이용하는 연구도 많이 수행되고 있다[4]. 본 논문에서는 이에 관한 모든 메카니즘들을 재검토한 후, CGI 애플리케이션의 성능을 향상시킬 수 있는 총체적인 구조를 제안하고자 한다. 그리고 제안한 CGI 애플리케이션 구조간의 성능분석을 한다.

본 논문의 구성은 다음과 같다. 제 2장에서 관련 연구들에 대해서 설명하고, 제 3장에서는 관련 CGI 애플리케이션 구조 및 제안구조, 제 4장에서 결론 및 향후 연구를 기술한다.

2. 관련 연구

2.1 CGI

CGI는 웹 서버와 서버측의 응용 프로그램 사이를 연결하기 위한 일종의 게이트웨이 표준으로써, 웹 서버로 하여금 사용자의 요구를 동적으로 수용할 수 있도록 하기 위해 만들어진 것이다.

그림 1은 CGI 실행 과정을 나타낸 것이다. 웹 브라우저에서는 URL이나 폼에 입력된 값을 실제 웹 서버에 전달할 때 요청헤드(Request Header)를 발생시켜 웹 서버에 전달한다.

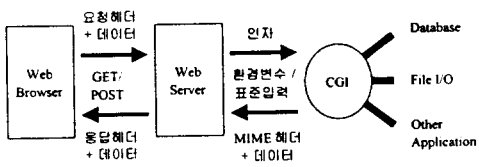

그림 1 CGI 실행 과정

CGI 프로그램은 웹 서버로부터 전달받은 인자 값을 지정된 형식으로 디코딩한 다음, 디코딩한 인자를 가지고 실제 원하는 처리를 수행한다. CGI 프로그램에 의해 처리된 결과를 MIME 헤더와 함께 웹 서버로 전달하게 되고 웹 서버는 CGI의 MIME에 따라 적절한 응답헤더를 생성시켜 웹 브라우저에 전달한다. 이러한 CGI는 다음과 같은 장점을 갖는다.

1) 단순성(simplicity) : 이해하기가 쉽다.

2) 언어 독립성(language independence) : CGI 애플리케이션은 어떠한 언어로도 작성될 수 있다.

3) 프로세스 고립성(process isolation) : 애플리케이션들이 분리된 여러 프로세스들로 동작하기 때문에, 버그있는 애플리케이션들이 웹 서버를 붕괴시키거나, 서버의 내부 상태를 액세스할 수가 없다.

4) 개방된 표준(open standard) : CGI 애플리케이션은 모든 웹 서버 상에서 구현될 수 있다.

5) 아키텍쳐 독립성(architecture independence) : CGI는 어떤 특별한 서버 아키텍쳐에 구속되지 않는다.

하지만, CGI 또한 다음과 같은 중요한 단점이 있다.

1) 성능(performance) : 각 요구마다 새로운 프로세스가 생성되고, 요구가 처리된 후 프로세스가 소멸되기 때문에 오버헤드가 커서 효율성이 낮다.

2) 제한된 기능 : CGI 애플리케이션은 단지 클라이언트의 요구에 대한 응답을 생성하는 응답자(responder) 역할만 한다. 그리하여 인증(authorization) 또는 로깅(logging)과 같은 웹 서버 요구 처리의 다른 기능들을 수행하지 못한다.

2.2 서버 API

CGI의 성능 문제를 개선하기 위해서 서버 개발자들이 그들 서버를 위한 API를 개발하게 되었다. 이들 중 대표적인 것으로 Netscape사의 NSAPI와 Microsoft사의 ISAPI가 있으며, 무료로 이용할 수 있는 Apache 서버도 또한 API를 제공하고 있다.

서버 API 방식에서는 애플리케이션이 서버 프로세스 내에서 동작하고, 프로세스는 매 요구들을 통해 영구적으로 동작하기 때문에, CGI 애플리케이션의 프로세스 생성 및 초기화 문제들을 개선할 수 있다. 그리하여 서버 API로 작성된 애플리케이션은 CGI 애플리케이션보다 크게 빨라질 수 있다.

또한 서버 API는 CGI보다 많은 기능을 제공할 수 있는데, 액세스 제어, 로그 파일 액세스, 서버 요구 처리의 다른 상태로의 연결 등의 서버 내부 동작들의 행위를 수행할 수도 있다. 하지만, 서버 API는 다음과 같은 문제점을 갖는다.

1) 복잡성(complexity) : 서버 API는 구현 및 유지보수 비용이 증가한다.

2) 언어 의존성(language dependence) : 애플리케이션들은 서버 API 의해 제공되는 언어(C/C++ 등)로 작성되어져야 한다. CGI 프로그램 작성을 위해 많이 쓰이는 Perl은 현존하는 어떤 서버 API와도 함께 사용될 수 없다.

3) 프로세스 고립성 결여 : 애플리케이션이 서버의 주소 공간에서 실행되기 때문에, 버그가 있거나 악의있는 애플리케이션이 서버를 붕괴시키거나 서버의 보안을 위협할 수 있다.

4) 독점성(Proprietary) : 특별한 API로 작성된 애플리케이션은 그 개발자의 서버에만 한정될 수 있다.

5) 서버 아키텍쳐 의존성 : API 애플리케이션들은 서버와 같은 아키텍쳐를 공유해야 한다. 만약 웹 서버가 다중스레드(multithread)로 구성된다면, 애플리케이션도 스레드에 안전(thread-safe)해야 한다. 또한 웹 서버가 단일 스레드 프로세스들을 갖는다면, 다중스레드 애플리케이션은 어떠한 성능 이득을 기대할 수 없게 된다. 서버 개발자가 서버 아키텍쳐를 변경하게 될 때, API도 보통 변경될 것이고, 애플리케이션들은 변경된 API로 적용되어지거나 재작성되어야 한다.

이 중 서버 API의 가장 큰 단점은 표준화 부재로 인한 게이트웨이 코드의 제한적인 이식성(portability)이다.

3. 관련 CGI 애플리케이션 구조 및 제안구조

3.1 CGI 애플리케이션 실행구조

동시에 많은 사용자들이 웹 서버에 서비스를 요구할 경우, CGI 애플리케이션의 실행구조에 따라 하나의 질의를 처리하기 위한 데이터베이스 연결 비용, 프로세스의 크기 및 프로세스 관리 비용이 달라지

고, 이는 전체 시스템의 성능에 영향을 준다. 본 논문에서는 CGI 애플리케이션의 실행구조를 CGI 실행모듈 방식과 CGI 서버 방식으로 분류한다.

3.1.1 CGI 실행모듈 방식

그림 2는 CGI 실행모듈 방식 또는 모놀리식(monolithic) CGI 방식의 실행구조를 나타낸다.

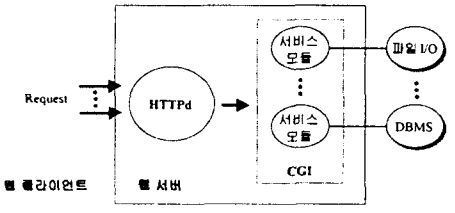

그림 2 CGI 실행모듈 방식

CGI 실행모듈 방식은 가장 일반적인 방식으로 프로세스의 생성 및 종료, 프로세스간의 자료 복사 및 교체 등으로 프로세스 관리 비용이 크다. 특히 동시에 많은 요구가 수행될 경우, 각 요구에 대한 CGI 프로세스가 매번 생성되고 그 때마다 데이터베이스 연결, 파일 I/O 등이 일어나게 된다. 그러므로 시스템 자원 부족 현상이 빨리 일어나게 되고 성능도 저하된다. 하지만 구현 및 시험이 간단하고, 이식성이 강한 장점이 있다.

3.1.2 CGI 서버 방식

CGI 서버 방식은 CGI 실행모듈 방식의 성능 문제를 해결하기 위해, CGI 애플리케이션 프로세스가 데몬으로 동작한다. CGI 서버 방식은 웹 서버의 CGI에 의해 구동되는 디스패처(dispatcher) 프로세스와 데이터베이스 연결 및 검색 등의 기능을 수행하는 서비스 모듈 데몬으로 나누어져 있다. 디스패처는 CGI용 실행화일로서 사용자의 요구시마다 새로이 실행되며, 웹 서버의 CGI로부터 프로세스 파이프로 연결되어 호출된다. 디스패처는 단지 사용자가 요구한 질의를 수행할 수 있는 서비스 모듈 데몬을 식별하여 사용자의 요구만을 전달하므로 프로세스의 크기가 매우 작고 데이타베이스와의 직접적인 접속이 일어나지 않는다. 그리하여 동시에 많은 요구가 발생해도 시스템 자원 부족 문제가 쉽게 발생하지 않는다. 서비스 모듈 데몬은 영구적으로 실행되고, 사용자의 요구를 처리하고 그 결과를 HTML 형식으로 디스패처에 반환한 후, 소멸되지 않고 다른 요구를 기다린다. 디스패처는 그 결과를 웹 서버에 전달한다. 그림 3은 CGI 서버 방식의 실행구조를 나타낸다.

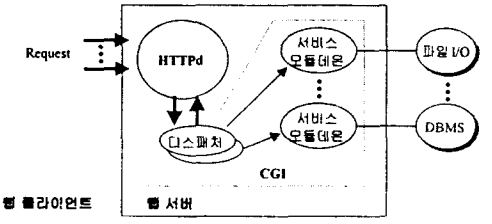

그림 3 CGI 서버 방식

CGI 서버 방식은 기본적으로 CGI 기술을 이용하기 때문에 CGI 실행모듈 방식의 장점을 그대로 갖는다. 아울러 CGI 실행모듈 방식에서의 여러 가지 성능 문제점을 해결하므로 본 논문에서 제안하는 CGI 애플리케이션 실행구조도 기본적으로 CGI 서버 방식을 이용한다.

3.1.3 CGI 서버의 구현

CGI 서버의 구현은 사용자 요구에 대한 처리를 프로세스와 스레드(thread)를 이용한 방식으로 나눌 수 있다.

하나의 프로세스는 주소 공간과 하나의 제어흐름을 가지며 이를 위해 페이지 테이블, 파일 기술자, I/O 요청과 레지스터 값 등의 많은 양의 상태정보를 초기화하고 관리해야 한다. 이러한 프로세스 관련 정보의 관리는 많은 비용이 들며 하나의 작업을 처리할 때 여러 프로세스를 사용한다면 동일한 자료가 중복되어 관리되어야 한다. 또한 프로그램이 실행될 때 하나의 실행흐름만이 가능하므로 병행처리가 어렵다는 단점을 가진다[5]. 반면 다중스레드는 하나의 프로세스 내에 다수의 실행흐름, 즉 스레드를 가질 수 있고 각각의 스레드가 힙(heap)과 정적자료 코드 부분을 공유한다. 또한 각각의 스레드는 관련 정보를 유지하기 위한 레지스터와 스택을 가진다. 이로 인해 자원의 생성 및 관리가 중복되는 것을 막을 수 있으며 서로 독립적인 수행이 가능해지므로 병행처리가 수월해진다[6].

다중스레드의 장점에는 응용 프로그램 반응 속도의 개선, 다중프로세서의 효율적 이용, 프로그램 구조의 명료화, 시스템 자원의 절약 및 성능 향상 등이 있다. 하지만 다중스레드는 프로그램 설계가 복잡하고, 스레드간의 동기화 문제(경쟁상태, 교착상

태, 우선 순위 역진 등)가 발생할 수 있으며, 라이브러리 형태로 제공되는 스레드 경우, 동일한 루틴에 대한 재진입(re-entrant)이 불가능한 단점이 있다[7]. 본 논문에서는 프로세스와 스레드를 이용한 CGI 서버의 구현방식을 다음과 같이 분류한다.

(1) 요구당 프로세스(process per request) 방식

가장 초기의 구현방식이고, 그림 4와 같이 서버 프로세스가 클라이언트의 요청을 받을 때마다 하나의 프로세스를 생성시켜(fork) 그 프로세스로 하여금 요청을 담당하게 한다. 초기의 WWW 서버 등이 CGI 프로그램을 사용자 요구마다 실행시키는 형태로서, CGI 프로그램의 실행 입장에서 보면 CGI 실행모듈 방식과 동일하다.

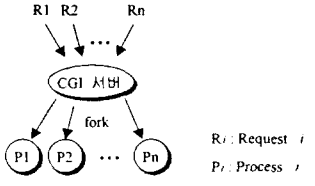

그림 4 요구당 프로세스 방식

(2) 프로세스 풀(process pool, pre-forking) 방식

프로세스의 생성 비용이 크기 때문에 많은 사용자 요구를 처리해야 하는 경우 문제가 된다. 그림 5와 같이 미리 정해진 개수의 프로세스를 생성한 다음, 하나의 요구가 처리 완료되더라도 해당 프로세스를 종료시키지 않고 계속 기다리게 하여 요청이 올 때마다 재사용하는 방식이다.

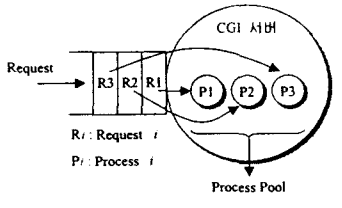

그림 5 프로세스 풀 방식

최근 운영체제에서 스레드를 지원하면서 점점 사용이 줄어들고 있다.

(3) 메시지 큐(queue)를 가진 단일 프로세스 서버

그림 6과 같이 사용자의 요구는 메시지 큐로 보내지고, 서버는 메시지 큐를 확인하고 각 요구를 처리 및 대기하는 방식으로, 프로세스 생성 및 관리 비용이 효율적이고, 문맥 교환(context switch) 시간 감소 등의 장점이 있다. 하지만 I/O가 많은 사용자 요구에 대해서 다중프로세서 방식보다 비효율적(blocking 때문)이고, 큐의 길이에 따라 사용자 수가 제한되는 단점이 있다.

그림 6 메시지 큐를 가진 단일 프로세스 서버

(4) 요구당 스레드(thread per request) 방식

그림 7과 같이 요구당 프로세스 방식과 유사하게 동작하며, 프로세스 생성 및 소멸시의 오버헤드를 줄이기 위해 요청을 받을 때마다 하나의 스레드를 생성시키는 방법이다.

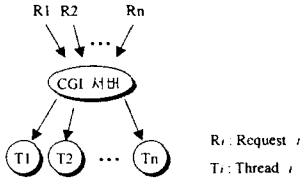

그림 7 요구당 스레드 방식

(5) 스레드 풀(thread pool) 방식

그림 8과 같이 프로세스 풀 방식과 유사하게 동작하며, 매 요청마다 스레드 생성 비용을 줄이기 위해 미리 정해진 수의 스레드를 생성한 후 새로운 요청이 올 때마다 할당하는 방식이다.

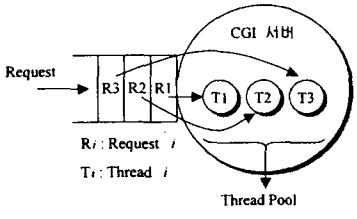

그림 8 스레드 풀 방식

이 방식은 서버가 사용하는 자원의 양을 제한할 수가 있으며, 요청이 들어왔을 때 사용할 수 있는 스레드가 없다면 큐에서 기다려야 한다.

(6) 세션당 스레드(thread per session) 방식

그림 9와 같이 매 요청마다 스레드 생성 비용을 줄이기 위해 사용하고, 여러개의 요청을 하는 클라이언트 하나마다 스레드를 할당한다. 이 방식은 하나의 클라이언트가 여러개의 요청을 오랜 시간에 걸

처서 하는 경우에 유용하다. 또한 전형적인 요구-응답방식의 웹 서비스에서 사용자 ID나 IP 등을 식별자로 해서 세션별로 클라이언트를 관리하게 되면, 효율적인 캐싱으로 성능을 향상시킬 수 있다.

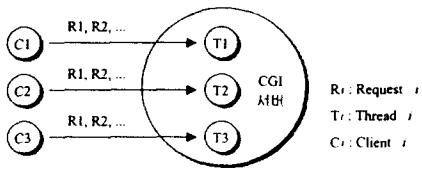

그림 9 세션당 스레드 방식

3.2 CGI 애플리케이션 실행구조 제안

제안하는 CGI 애플리케이션 실행구조는 CGI와 서버 API의 장점을 조합한 구조이다. 기본적으로 CGI 서버 모듈로 구성되며, 각 모듈은 세션당 스레드 방식을 적용한다. 이것은 클라이언트별 세션을 유지함으로 인해 캐쉬의 적중율을 높이고, 전체적인 성능을 향상시킬 수 있다. 또한 CGI 서버 모듈은 독립적인 프로세스로 실행되므로 버그있는 애플리케이션으로 인한 서버 및 다른 애플리케이션의 붕괴, 서버의 보안정보(암호에 대한 세션키 등) 액세스 등의 위험이 없다. 또한 부하 분산과 외부 웹 사이트들의 관리에 유용한 원격지 애플리케이션 실행 기능을 제공한다. 이러한 분산 컴퓨팅은 현존하는 legacy 시스템의 확장성, 연결성, 시스템 이용률 향상, 파이어월을 통한 보안 향상 등을 지원한다. 기본적으로 웹 서버와 애플리케이션 사이에 TCP 연결을 통해서 원격의 시스템상에서 동작이 가능하게 된다.

3.2.1 요구 처리 절차

다음은 제안하는 CGI 애플리케이션 실행구조에서의 요구 처리 절차를 나타낸다.

1. 웹 서버는 요구들을 처리하기 위한 CGI 서버 프로세스들을 생성한다. 이 CGI 서버는 시스템 시작시 또는 요구시마다 생성되어질 수 있다.

2. CGI 서버 모듈은 초기화되고 웹 서버로부터 새로운 연결을 기다린다.

3. 클라이언트의 요구가 도착했을 때, 서버는 디스패처를 실행하여, CGI 서버 모듈에 대한 연결을 설정하고, 이 연결을 통해서 CGI 환경변수 정보와 표준입력을 전송한다.

4. CGI 서버 모듈은 설정된 연결을 통해 표준출력과 에러정보를 서버에게 전송한다.

5. CGI 서버 모듈이 연결을 닫을 때, 클라이언트 요구가 처리 완료된다. 그 후에도 CGI 서버 모듈은 웹 서버로부터의 다른 연결을 위해 대기한다.

3.2.2 캐싱

디스크로부터 정보를 읽어오는 것은 캐쉬에서 읽어오는 것보다 더많은 프로세스 시간을 요구하기 때문에, 캐싱은 디스크 I/O와 프로세스 시간 모두를 줄일 수 있다. CGI 프로세스는 하나의 요구를 처리한 후 소멸되기 때문에 메모리상에 캐싱을 할 수 없지만, 서버 API는 이 문제를 해결한다. 대부분의 서버 API는 프로세스 풀 서버 모델이며, 이것은 부모 프로세스와 자식 프로세스 풀로 구성된다. 하지만, 각 프로세스들은 메모리를 공유하지 않고 요구가 도착하면 랜덤하게 자식 프로세스를 할당하므로, 캐쉬의 효율성이 떨어진다. 예를 들면, 자주 사용되는 파일을 메모리상에 두기 위해 서버는 자식 프로세스마다 복사본을 가지므로 메모리 낭비가 심하다. 또 이 파일이 수정되면 복사본을 가진 모든 자식 프로세스에게 파일 수정을 알려야 함으로 복잡하다. 본 논문에서 제안하는 CGI 애플리케이션 실행구조는 클라이언트의 요구들은 세션당 스레드로 동작되므로, 서버 API 보다 캐쉬 적중률이 높다.

3.3 성능분석

본 논문에서 제안하는 CGI 애플리케이션 실행구조에 대한 성능을 분석하기 위해 다음과 같은 가정을 둔다.

1. 멀티프로세싱과 멀티스레딩 구조에서 요구 처리시간 내에 문맥교환이 일어날 확률은 P이다.

2. 데이터베이스에서의 처리시간 T_{DB}는 동일하다.

3. 주어진 요구의 프로세스 또는 스레드에서의 처리시간(디코딩 등)은 무시할 만큼 작다.

4. CGI 프로세스와 다른 애플리케이션 사이의 요구의 전파시간 d는 일정하다.

본 논문에서의 이러한 가정은

성능 분석을 위해 프로세스와 스레드의 생성 및 동기화시의 지연을 나타내는 표 1의 값을 이용한다.

표 1 SPARCstation 2에서의 user thread, LWP, 프로세스 동작 지연[8]

	생성시간 (msec)	동기화시간 (msec)
User thread	52	66
LWP	350	390
Process	1700	200

3.3.1 요구당 프로세스 또는 스레드 방식

그림 4와 7에서 프로세스의 생성시간을 T_{pc}라 하면, 스레드의 생성시간 T_{tc} = (52/1700)T_{pc} = 0.03T_{pc}가 된다 또 확률 P로 동기화가 발생할 때, 프로세스 동기화 시간을 Tps라 하면, 이에 대해 스레드 동기화 시간 T_{ts} = (66/200)T_{ps} = 0.33T_{ps}이다. 그림 10은 요구당 프로세스 방식에서 각 프로세스의 처리시간을 나타낸다.

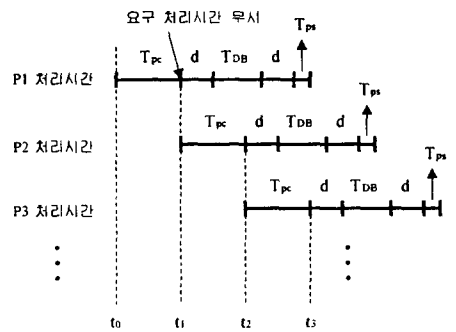

그림 10 요구당 프로세스의 각 프로세스 처리시간

그림 10에서 각 프로세스들의 처리시간은 다음과 같은데, 2d+T_{DB}은 일정하므로 상수 C로 둔다.

P1의 처리시간 = T_{pc}+2d+T_{DB}+T_{ps} = T_{pc}+T_{ps}+C

P2의 처리시간 = 2T_{pc}+2d+T_{DB}+T_{ps} = 2T_{pc}+T_{ps}+C

. . .

Pn의 처리시간 = nT_{pc}+2d+T_{DB}+T_{ps} = nT_{pc}+T_{ps}+C

이를 요구당 스레드 방식에 적용하면, n개의 스레드 처리시간 Tn = 0.03nT_{pc}+0.33T_{ps}+C가 된다.

3.3.2 프로세스 풀과 스레드 풀 방식

프로세스 풀과 스레드 풀 방식의 성능 분석 모델은 큐잉이론의 적용이 필요하며, 현재 평가식에 관해 연구중이다. 이외 메시지 큐를 이용한 단일 프로세스 서버 방식과 세션당 스레드 방식 또한 성능 분석 모델과 평가식도 연구중이다.

3.3.3 메모리 요구량

CGI 애플리케이션 실행 구조에서 사용자 요구에 대한 프로세스 처리시간은 사용자의 응답시간과 관련이 있으며, 메모리 요구량은 서버에서 동시에 처리할 수 있는 사용자 수에 직/간접적으로 관계된다. CGI 애플리케이션 실행 구조의 메모리 요구량은 그림 11과 같은 순서로 예상된다.

메시지 큐를 이용한 단일 프로세스 서버 < 스레드 풀 방식 < 프로세스 풀 방식 < 세션당 스레드 < 요구당 스레드 방식 < 요구당 프로세스 방식

그림 11 메모리 요구량 비교

4. 결론 및 향후 연구

웹의 폭넓은 사용은 동적 웹 컨텐트를 생성하는 CGI 애플리케이션에 대한 많은 요구를 발생시켰다. 사용자 요구가 집중될 때, 기존의 CGI 애플리케이션들은 전체 시스템의 주요한 성능 저하 요인이 된다. 본 논문에서는 이러한 CGI의 단점을 극복하기 위하여 개선된 CGI 애플리케이션 실행구조를 제안하였으며, 이와 관련한 성능을 분석했다.

향후의 연구과제로는 프로세스 풀과 스레드 풀 방식과 메시지 큐를 이용한 단일 프로세스 서버 방식 및 세션당 스레드 방식의 성능 분석 모델과 이의 평가식 유도에 관한 연구가 있다.

참 고 문 헌

[1] W.E.Weinman. "The CGI Book", New Riders, 1996.

[2] M. F. Arlit and C. L. Williamson, "Web Server Workload Characterization: The Search for Invariants", Proc. of SIGMETRICS96, ACM, Philadephia , May, 1996.

[3] 백승구, 임인택, 김수정, 천성권, 김종근, "웹과 DB연동시 CGI모델과 자바모델의 성능 평가", 한국정보과학회, '98 봄 학술발표논문집, 제 25권, 제 1호, pp.328-330.

[4] 최용준, 임경수, 황도삼, 김종근, "계층적 정보구조의 web시스템 관리기술", 한국정보처리학회 논문지, 제 5권 5호 게재 예정.

[5] 노경현, 설승진, 이금석, "Solaris, POSIX, Java 다중스레드 모델의 성능 비교", 한국정보처리학회, '97 춘계 학술발표논문집, 제4권 제1호, pp.177-182, 1997

[6] A. S. Tanenbaum. "Distributed Operating Systems", Prentice Hall, 1992.

[7] D. Stein, D.Shah, "Implementing Lightweight Threads", Summer '92 USENIX Conference, pp.1-9, 1992

[8] Vahalia, U. "UNIX Internals : the new frontiers", Prentice Hall, 1996.