제 19 회 한국정보처리학회 춘계학술발표대회 논문집 제 10 권 제 1 호 (2003. 5)

WebLogic-Tuxedo 기반 시스템의 성능 분석

김인규, 최원용
국민대학교 비즈니스컴퓨팅 연구실
e-mail : inkim@kookmin.ac.kr

Performance Analysis of WebLogic-Tuxedo Based System

In-Kyu Kim, Won-Yong Choi
Business Computing, Kook-Min University

요 약

인터넷의 확산과 컴퓨터 기술의 발전으로, 기존에 인트라넷 환경에서 사용되던 정보 처리 시스템을 인터넷 환경에서 사용할 수 있게 하려는 노력들이 활발히 진행되고 있다. 그리고 그에 따른 수많은 대안들이 나타났으나 이런 대안들의 성능에 대해서는 신뢰성을 갖기에 부족함이 많았다.

본 논문에서는 인터넷으로 확장된 여러 시스템 중 WebLogic-Tuxedo 기반 시스템을 선정하여, 인터넷으로의 통합 시스템을 구현하고 그 성능을 측정하였다.

1. 서론

본 연구에서 레거시 시스템(Legacy System)이라 지칭하는 시스템은 인터넷을 통한 정보처리를 지원하지 않는 전통적인 클라이언트/서버시스템(Client/Server System)이다. 또, WebLogic-Tuxedo 기반 시스템이라 지칭하는 시스템은 WAS(Web Application Server)로 WebLogic 을 사용하고, TP Monitor(Transaction Processing Monitor)로 Tuxedo 를 사용하며, WebLogic-Tuxedo 간 연결을 위해 Jolt 를 사용하는 시스템이다.

인터넷의 성장과 보편화로 인해 기존에 지역 망에 의존하던 레거시 시스템들은 인터넷으로 나타낼 수 있는 형태를 지원하기 위해 여러 가지 방법으로 재개발되거나, 또는 새로운 인터넷 환경으로의 통합이 진행되고 있다. 이런 기존 레거시 시스템의 재개발, 통합과정으로 만들어진 전체 시스템은 안정성이나 성능에 대한 불확실성을 내포하고 있어, 이에 대한 검증이 필요하다.

본 연구에서는 레거시 시스템과의 통합이나 재개발 과정에서 시스템의 성능분석에 대한 객관적이고 공개적인 연구결과를 제공하는 것을 목적으로 한다. 레거시 시스템의 구체적인 예로 현재 보편적으로 사용되고 있는 TP Monitor 제품인 BEA 사의 Tuxedo 를 지정하고, 레거시 시스템의 통합 역할에는 EJB(Enterprise Java Beans)를, 인터넷 표현 부분에 BEA 사의 WebLogic 을 설정해 WebLogic-Tuxedo 기반의 통합 시스템을 구성한다.

2. 레거시 시스템 환경의 변화

클라이언트/서버 위주의 레거시 시스템들은 변화하는 인터넷 환경에 적응하기 위해 신규개발과 통합이라는 두 가지 방법으로 인터넷 환경에 적응하려 하고 있다. 기존 레거시 시스템을 인터넷 환경에서 사용할 수 있는 두 가지 대안을 예로 들면, 첫째 현존하는 레거시 시스템을 전체적으로 분석하여 새로운 시스템으로 완전히 진화 시키는 방법과 둘째 Tuxedo 나 TMAX 등 미들웨어를 사용하여 시스템의 외부 접촉 점만을 소규모로 변경하여 통합 시키는 방법이 있다. 위의 두 대안을 각각 비교해 볼 때 첫 대안은 레거시 시스템을 현대화하기 위해 가장 좋은 방안이지만 이미 상당한 투자가 행해진 레거시 시스템을 새로 개발하는 것은 레거시 시스템 개발 및 유지보수를 위해 사용된 비용을 다시 발생시키기 때문에 경제적인 관점으로 볼 때, 미들웨어를 이용하여 레거시 시스템을 현대화하는 두 번째 대안이 비용-효율적인 방법이다. 두 번째 대안을 사용함으로써 레거시 시스템을 보호하고 지원할 수 있으며, 레거시 시스템을 중지 없이 사용할 수 있고, 레거시 시스템의 진화를 촉진할 수 있

다.[1]

3. 통합을 위한 대안

새로운 인터넷 환경으로 레거시 시스템을 통합하기 위한 시도는 대부분이 미들웨어를 이용해 이루어지고 있다. 미들웨어는 클라이언트와 서버간의 상호작용을 지원하기 위해 필요한 모든 분산 소프트웨어를 통칭하는 광의의 개념이다.[3]

대표적인 통합 방법들로는 CORBA(Common Object Request Broker Architecture)를 이용한 통합 방법과 각 미들웨어(BEA 사 Jolt, Tmax 사 WebT) 공급 사들이 제공하는 소프트웨어를 이용한 통합 방법이 있다.

3.1 통합대안 비교

TP Monitor 는 트랜잭션이 프로세스(Process) 내의 한 단계로부터 다음단계로 잘 넘어가는지를 감시하는 프로그램이다. CORBA 를 이용한 통합은 과거 여러 종류의 TP Monitor 사용여부에 관계없이 적용할 수 있다는 장점이 있는데 비하여 미들웨어 공급 사들이 제공하는 제품을 이용한 통합은 각 미들웨어 공급사의 TP Monitor 를 사용하고 있는 경우에만 적용할 수 있다는 단점이 있다. 그러나 CORBA 를 이용한 통합 시에도 VisiBroker 나 Orbix 등 ORB(Object Request Broker) 제품을 구입해야 한다는 점과 미들웨어 공급 사들의 제품군을 이용한 통합방법이 비교적 용이하게 적용할 수 있다는 점을 고려해 볼 때 미들웨어 공급 사들이 제공하는 제품을 이용한 통합방법 역시 의미를 가진다. 따라서 본 연구에서는 미들웨어 공급 사들이 제공하는 제품을 이용한 통합방법에 의하여 인터넷 환경과 레거시 시스템의 통합을 시도하였으며, 그 중 웹 환경을 통한 정보처리를 지원하지 않는 전통적인 클라이언트/서버 시스템에서 가장 많이 사용된 BEA 사의 미들웨어 제품군(WebLogic, Tuxedo)을 이용하여 시스템을 구현하였다.

4. 연구방법

4.1 연구목적

인터넷 환경을 위해 레거시 시스템을 완전히 새로운 시스템으로 다시 개발하는 것은 과거의 시스템을 이해하기 위해 추가 노력이 필요하고, 요구사항의 만족비율이 크게 떨어지는 단점이 있다. 그러나 미들웨어를 이용한 통합은 레거시 시스템을 보호하고 지원하며 레거시 시스템을 중지 없이 사용할 수 있게 하고, 진화를 촉진하는 장점이 있다. 그러므로, 본 연구에서는 통합을 위한 대안중의 하나인 WebLogic-Tuxedo 기반 시스템을 구현하고 그 성능을 측정,평가하여 인터넷 환경과 레거시 시스템의 통합을 위한 Solution 을 제공한다.

4.2 기존연구에서의 연구방법

본 연구에서는 WebLogic-Tuxedo 기반 시스템의 성능을 측정하고 평가하기 위해 Juric 등의 기존연구를 참고하였다. Juric 등은 그들의 연구에서 CORBA(Common Object Request Broker)와 RMI(Remote Method Invocation)의 성능을 비교하였다. 그들은 CORBA 와 RMI 에서 발생하는 과부하(Overhead)를 측정하는 것이 성능의 비교척도가 될 수 있다고 생각하여 성능을 측정하기 위하여 메소드 호출 시에 발생하는 CORBA 와 RMI 의 과부하를 비교하였는데 과부하 테스트를 위해서 다음과 같은 선행조건이 만족되어야 한다고 생각하였다.

- 조건 1: RMI 테스트 결과와 CORBA 테스트 결과의 비교가 가능한 것이어야 함.
- 조건 2: 단일 클라이언트와 복수 클라이언트 시나리오를 시뮬레이션 해야 함.
- 조건 3: 여러 유형 자료형(Data Type)을 사용해야 함.
- 조건 4: 전형적인 사용자환경에서 이용되어지는 하드웨어 장비를 사용해야 함.

4.3 연구방법

본 연구에서도 과부하(Overhead)를 측정하는 것이 성능의 비교척도가 될 수 있다는 Juric 등의 연구에서 사용된 기본 가정을 이용하였다. WebLogic-Tuxedo 기반 시스템의 과부하 측정을 통하여 성능을 측정하고자 하였으며, Juric 등의 연구에서 성능비교를 위해 만족시키고자 했던 네 가지 선행조건을 고려하여 테스트하였다.

- 단일 클라이언트와 복수 클라이언트 시나리오를 시뮬레이션:
  1)웹클라이언트(인터넷 연결을 테스트하는 프로그램)와 웹어플리케이션(JSP), 레거시 어플리케이션(Tuxedo Service)을 각각 독립적인 컴퓨터에 위치시켜, 비교 가능한 테스트 조건을 구성하였다.
  2)동시 접속자의 숫자 증가에 따른 변화를 측정하기 위하여 웹클라이언트 테스트 프로그램은 복수개의 Thread 를 생성하였으며, 각 Thread 에서 웹어플리케이션으로 요청을 전송하였다.
- 인터넷에 HTML 로 표현하기 위한 자료형(Data Type)은 문자열(String)만으로 충분하다는 가정하에 테스트 데이터로 문자열만 사용하였다.
- 외부의 영향으로 인한 Traffic 의 발생이 결과에 영향을 주지않기 위해 테스트에 사용되는 기기들을 동일 Hub 를 이용하여 연결함으로써 테스트환경 외부로부터의 Traffic 의 영향을 최소화하였다.

성능분석의 평가를 위하여 기본적인 측정단위를 설정하였다.

- 웹클라이언트에서 웹어플리케이션(jsp)을 호출한 후 응답을 받을 때까지의 시간: t1
- 웹어플리케이션(jsp)에서 소요된 총시간: t2
- 웹어플리케이션(jsp)에서 Tuxedo Service 를 호출한 후 응답을 받을 때까지의 시간: t3
- Tuxedo Service 에서 소요된 총시간: t4

성능분석을 위하여 본 연구에서 평가하고자 한 항목은 웹클라이언트에서의 응답시간을 TTot, 웹클라이언트와 웹어플리케이션간의 통신시간을 TWeb, 웹어플리케이션과 Tuxedo Service 간의 통신시간을 TTux 로 정의하면 다음과 같은 수식이 성립한다.

- TTot = t1 : 웹클라이언트에서 웹어플리케이션를 거쳐 Tuxedo Service 를 호출한 전체 응답시간을 측정한 시간.
- TWeb = t1 - t2 : 웹클라이언트와 웹어플리케이션간의 통신시간.
- TTux = t3 - t4 : 웹어플리케이션과 Tuxedo Service 간의 통신시간.

5. 시스템 개발환경 구성 및 구현

WebLogic-Tuxedo 기반 시스템은 WAS(Web Application Server)로 WebLogic 을 사용하고, TP Monitor 로 Tuxedo 를 사용하며, WebLogic-Tuxedo 간 연결을 위해 Jolt 를 사용하는 시스템으로 웹어플리케이션인 JSP(Java Server Page) 어플리케이션, Jolt 세션(Session)을 관리하는 EJB 어플리케이션, 레거시 어플리케이션인 Tuxedo Service 로 구성되어진다.('그림 1')

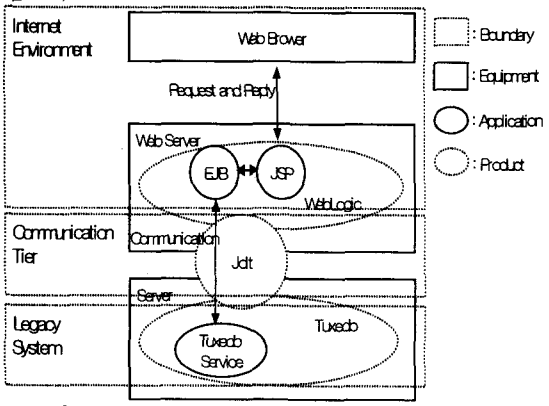


그림 1 WebLogic-Tuxedo 기반 시스템 구성

5.1 Tuxedo 환경구성 및 Service 개발

본 연구에서 레거시 시스템으로 가정된 Tuxedo Service 는 FML32 형의 Buffer 를 사용하는 형태로 개발되었으며, 다음순서에 따라 구현되었다.

1) 사용자 설정파일에 Tuxedo 관련 Parameter 추가
2) Tuxedo 환경설정
3) 입출력 Buffer 작성
4) Tuxedo Service 개발
5) Jolt 관련 환경설정
6) Jolt Repository 에 작성

ubbconfig 라는 이름으로 Tuxedo 환경설정 파일을 작성하였다. ubbconfig 에 기록된 Tuxedo 환경관련 설정은 Tuxedo 서버그룹 설정, Tuxedo 서버설정, Tuxedo Service 관련 설정, Tuxedo Listener 관련 설정 등이다. Tuxedo Listener(WSL) 환경은 Handler(WSH)를 최소 20 개, 최대 100 개까지 생성하고 클라이언트의 요청을 최대 20 개까지 처리하도록 설정하였다. 작성된 Tuxedo 환경설정은 Tuxedo 관리명령인 tmloadcf 를 이용하여 Load 하였다.

Tuxedo Service 의 Buffer 설정파일을 작성하고 mkfldhdr32 명령을 이용하여(Buffer Type 이 FML 이라면 mkfldhdr 명령을 사용한다.) 개발될 Tuxedo Service 의 입출력 Buffer 를 작성하였고, 작성된 Tuxedo Service Buffer 에 대한 정보의 위치를 ubbconfig 파일에 추가하였다.

Tuxedo Service 를 작성하고 관련설정을 ubbconfig 파일에 추가하였다. Tuxedo Service 가 실행되기 시작된 시점과 종료되기 직전의 시점에 clock() 함수를 이용하여 시간을 측정하고 차이를 계산하여 Tuxedo Service 에서 소요된 총시간(t4)을 산출하도록 하였다.

Tuxedo Service 와 인터넷 환경과의 연결역할을 수행하는 BEA 사의 제품인 Jolt 사용을 위하여 ubbconfig 파일 내에 Jolt Server Listener(JSL) 환경을 설정하였다.

Jolt Repository 에 Tuxedo Service 를 등록하기 위하여 개발한 Service 명, Service 의 Buffer Type, Buffer 내의 Field 정보 등을 jtservicefile 이라는 이름의 파일에 기록하고 BEA Jolt 유틸리티인 Bulk Loader 를 이용하여 호출될 Tuxedo Service 의 정보를 Load 하였다.

5.2 WebLogic 환경구성 및 웹어플리케이션 개발

EJB Component 를 개발하여 Tuxedo Service 의 호출을 담당하도록 하였으며, 웹어플리케이션은 JSP(Java Server Page)를 이용하여 개발하였다.

1) WebLogic 환경설정
2) EJB 컴포넌트 개발
3) XML 파일 작성
4) 웹어플리케이션 개발
5) Jolt 관련 환경설정 추가

EJB 컴포넌트는 Jolt Session 관리기능을 Business Logic 과 분리하기 위하여 사용되었다. EJB 컴포넌트를 작성함으로써 웹어플리케이션은 Business Logic 만을 구현하고 EJB 를 통하여 Tuxedo Service 를 호출한다. EJB Contatiner 역할은 WebLogic 이 수행한다. 개발된 EJB 컴포넌트들은 다음과 같다.

- JoltTest interface: EJBObject Interface 상속.
- JoltTestHome interface: EJBHome Interface 상속.
- JoltTestBean class: SessionBean 을 구현하고 Tuxedo Service 를 호출하기 위한 Method 를 작성.

SessionBean Interface 는 Business Logic 을 수행하는 Object 를 구현하는데 사용된다. Tuxedo Service 호출 전과 후에 System.currentTimeMillis() method 를 이용하여 시간을 측정하고 차이를 계산함으로써 웹어플리케이션에서 Tuxedo Service 를 호출한 후 Tuxedo Service 로부터 응답을 받을 때까지의 시간(t3)을 산출하였다.

Tuxedo Service 와 웹 환경과의 연결역할을 수행하는 BEA 사의 제품인 Jolt 사용을 위하여 Web 서버가 실행되는 시스템에서 config.xml 파일에 WebLogic Server 와 Tuxedo Domain 의 JSL(Jolt Server Listener) 사이에 위치하는 Jolt Connection Pool 이 최소 20 개에서 최대 40 개까지 생성되도록 설정하였다.

5.3 측정방법 및 측정프로그램 작성

성능의 측정을 위해 4.3 에서 설정한 TTot, TWeb, TTux 를 산출하기 위하여 테스트 프로그램이 작성되어 WebLogic-Tuxedo 기반 시스템의 성능을 시뮬레이션(Simulation)하였다.

- ClientConnetion.java: Jolt 서비스를 호출하는 jsp 를 실행하는 기능, 동시요청을 시뮬레이션 하기 위해 Thread Class 를 상속하여 입력된 실행 개수에 따라 thread 로 반복 실행. 웹어플리케이션에 요청을 전송하기 위해 사용된 Class 는 URL 과 URLConnection 이고, 응답을 읽어 들이기 위하여 사용된 Class 는 DataInputStream 와 BufferedInputStream 이다.
- Client.java: ClientConnetion 의 instance 인 thread 를 입력 변수에 따라 실행하고 계산된 TTot, TWeb, TTux 의 평균값을 최종 결과 파일에 저장.
- FileManager.java : 파일에 자료 저장.
- TestVO.java : 각종 테스트 데이터의 형태에 따라 데이터 처리의 일관성을 유지하기 위해 사용.

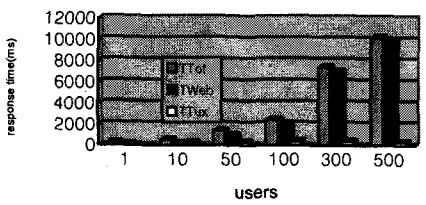
JDK 의 System.currentTimeMillis() Method 를 웹어플리케이션(JSP)을 호출한 후 응답을 받을 때까지의 시간(t1)을 측정하기 위해 사용하였다. 웹어플리케이션에서 소요된 총시간(t2)과 웹어플리케이션에서 Tuxedo Service 를 호출한 후 응답을 받을 때까지의 시간(t3)은 웹어플리케이션으로 부터 전송 받았고, Tuxedo Service 에서 소요된 총시간(t4)은 웹어플리케이션을 경유하여 Tuxedo Service 로부터 전송 받았다.

6. 성능 측정 결과

TTot: 웹클라이언트에서 웹어플리케이션를 거쳐 Tuxedo Service 를 호출한 응답시간을 테스트 프로그램에서 측정한 시간.

TTux : 웹어플리케이션와 Tuxedo Service 간의 통신시간.

TWeb : 웹클라이언트와 웹어플리케이션간의 통신시간.



(그림 2) 10KB 문자열

7. 결론

인터넷을 통한 정보처리 환경과 전통적인 클라이언트/서버 시스템을 통합할 수 있는 여러 솔루션 중 현재 보편적으로 사용되고 있는 WebLogic-Tuxedo 기반 시스템이 선택될 가능성이 높다. 따라서 WebLogic-Tuxedo 기반 시스템을 구현하고 성능을 측정/평가하였다.

WebLogic-Tuxedo 기반 시스템 성능 측정결과는 6 장의 그래프가 보여주는 것과 같이 동시요청의 개수(대략 thread 100 개)가 증가함에 따라 웹클라이언트와 웹어플리케이션간의 통신시간(TWeb)이 급격히 증가하는 현상이 발생했다. 그러나 Web Server 와 Tuxedo Service 간의 통신시간(TTux)은 큰 변화를 보이지 않았다. 따라서 WebLogic-Tuxedo 기반 시스템의 전체 수행시간에 영향을 미치는 것은 웹클라이언트와 웹어플리케이션간의 통신시간(TWeb)이다. 웹클라이언트와 웹어플리케이션간의 통신시간(TWeb)을 개선하기 위해서는 동시 사용자 수가 대략 50 ~ 100 명이 넘을 경우 운영시스템의 사양을 높여야 할 것 같다.(운영시스템에 따라 유동적임)

본 연구에서는 과부하(응답시간) 측정을 통하여 WebLogic-Tuxedo 기반 시스템의 성능을 측정하고 평가하였으나 전송률이나 에러발생률 등 다른 항목까지 고려하여 연구를 수행한다면 WebLogic-Tuxedo 기반 시스템의 성능 측정을 보다 정확하게 할 수 있을 것이다.

참고문헌

[1] Chia-Chu Chiang, "Wrapping Legacy Systems for Use in Heterogenous Computing Environments", Information and Software Technology 43 (2001) 497-507

[2] Greg Simco, "The Internet 2 Middleware Initiative", Internet and Higher Education 4 (2001) 77-84.

[3] Robert Orfali, Dan Harkey, Jeri Edwards, "Client/Server Survival Guide", Wiley, 1999

[4] Jeri Edwards, "3-Tier Client/Server At Work, Wiley", 1999.

[5] Nicholas Kassem and the Enterprise Team, "Designing Enterprise Applications with the JavaTM2 Platform", Enterprise Edition, Sun Microsystems, Inc. Oct 3, 2000.

[6] F. De Truck, S. Vanhastel, B. Volckaert, P. Demeester, "A Generic Middleware-based Platform for Scalable Cluster Computing", Future Generation Computer Systems 18 (2002) 549-560.

[7] Gerardo Canfora, Aniello Cimitile, Andrea De Lucia, Giuseppe A. Di Lucca, "Decomposing Legacy Programs: A First Step Towards Migrating to Client-Server Platforms", The Journal of Systems and Software 54 (2000) 99-110.

[8] H. M. Deitel, P. J. Deitel, "JavaTM How To Program", Prentice Hall, 1999.

[9] M.B. Juric, I. Rowman, M. Hericko, "Performance Comparison of CORBA and RMI", Information and Software Technology 42 (2000) 915-933, pp. 915