Journal of Korea Multimedia Society Vol. 18, No. 12, December 2015(pp. 1501-1508)
http://dx.doi.org/10.9717/kmms.2015.18.12.1501

부하테스트를 이용한 금융 시스템의 성능개선 사례

김태석[†], 이종윤[††], 김종수[†††]

A Case Study for Improving Performance of A Banking System Using Load Test

Tai Suk Kim[†], Jong Yun Lee[††], Jong Soo Kim[†††]

ABSTRACT

In this paper, we describe a case study to improve performance through the load testing of multi-tired system for financial accounts before the system opening. The load test was conducted after the data collection tools(Performance Monitor, DB PSSDiag) were installed. By analyzing the collected log, we were able to identify the main sector requiring performance improvements among the presentation tier, web tier, business logic tier and data tier. The ASP.NET server-down on the web tier could be improved by modifying the parameter values in the configuration file. Some server downs occurred on the business logic tier when a large number of users access at the same time, were more difficult to be solved. By analyzing the hang-dump at the server-down time, we were able to find a process that caused the problem. and we had to modify the relevant codes. For major performance improvements of the data-tier, indices of some queries was optimized by using the built-in DBMS query analyzer, after analyzing the log of long-response-time queries. The problems and solutions considered in this case study will be a reference for the performance improvement of a multi-layer system with the similar structure.

Key words: Load Test, Multi Tier, Server, Performance, Counter

1. 서 론

일반적으로 은행에서 사용되는 종합전산시스템은 금융거래를 처리하는 계정계 시스템, 통계나 상품개발 등을 담당하는 정보계 시스템 그리고 타 금융기관, 인터넷 기타 네트워크를 이용하여 외부기관과 연결을 담당하는 대외계 시스템 등으로 나누어진다. 이중에서 은행의 주요 업무를 처리하는 계정계 시스템은 그 특성상 전산으로 입력되는 데이터에 대해서는 어떠한 시스템 장애가 발생되더라도 반드시 정상적으로 처리되어야 한다는 특징이 있으며, 또한 해당정보의 빠른 처리도 주요 요건이다. 따라서 사용자들이 시스템을 실질적으로 사용하게 되는 실가동 이전에 장애에 대한 충분한 테스트가 필요하다.

다수 클라이언트들의 요구사항을 장애 없이 실시간으로 처리하기 위해서는 많은 수의 서버 군으로 구성되는 다계층 시스템이 필요하며, 이러한 경우 시스템을 구성하는 다양한 하드웨어와 소프트웨어가 성능 저하를 일으키는 원인이 될 수 있으며, 이는 전체시스템의 성능 저하를 초래한다. 본 논문에서는 금융권에서 구축된 전산시스템을 대상으로 다수의 클라이언트들이 해당 시스템을 실제로 사용하기 전에

※ Corresponding Author : Tai Suk Kim, Address: Dept. of Computer Software Engineering, Dong-eui University 176 Eomgwangno Busan_jin_gu, Busan 614-714, Korea, TEL : +82-51-890-1707, FAX : +82-51-890-1724, E-mail : tskim@deu.ac.kr
Receipt date : Aug. 5, 2015, Revision date : Oct. 19, 2015
Approval date : Oct. 21, 2015

[†] Dept. of Computer Software Engineering, Dong-Eui University
[††] Dept. of Computer Software Engineering, Dong-Eui University
joylee@microsoft.com
[†††] Dept. of System Management, Korea Lift College
seatree@klc.ac.kr

장애를 미리 진단하기 위한 부하테스트 수행과 부하테스트 후에 수집된 로그를 사용하여 성능 개선이 필요한 부분의 성능을 개선하는 사례를 소개한다.

2. 관련 연구

2.1 주요 계층 간 통신 아키텍처

전산시스템의 성능 개선과 관련된 주요 연구로는 비용이 많이 소요되는 전산시스템의 부하 테스트를 통계적 회귀분석을 이용한 추정 기법으로 성능을 예측한 연구[1], 상용화된 사례를 통하여 시간이 많이 소모되는 추상적 부하 테스트를 자동으로 접근하여 결과를 확인하기 위한 연구[2], 증가하는 클라이언트들과 더욱 복잡해진 요구사항을 수용해야 하는 대용량 웹 사이트의 부하 테스트 방법에 대한 연구[3-4], 대규모 온라인 게임 플레이어들을 수용하기 위한 서버 설계에 대한 연구[5], 공정의 효율을 개선하기 위해 센서로부터 수집된 빅데이터를 효율적으로 처리하기 위한 방법에 대한 연구들이 있다[6]. 본 논문에서 다루고 있는 주요 주제인 윈도우즈 서버를 이용한 다계층 전산시스템의 성능 개선 방법과 관련해서는 많은 온라인 또는 오프라인 문헌이 잘 정리되어 있으며, 다계층 전산 시스템 구축에 유용한 참고 자료로 사용되고 있다[7-14]. 본 논문에서 고려하고 있는 시스템 환경인 금융권에서 구축되는 계정계 전산 시스템에서 주요 계층인 비즈니스 로직과 웹 계층, 프레젠테이션 계층 간의 통신 아키텍처의 예를 Fig. 1에 나타냈다.

비즈니스 로직 계층의 주요 역할을 담당하는 서버들이 생성하는 쓰레드들은 COM 객체를 통하여 웹 서버의 Client COM 객체와 통신하며, D-MQ, S-MQ를 통하여 프레젠테이션 계층의 단말에 있는 NavClien COM과 AlertBar 객체와 통신한다. 웹 계층과 프레젠테이션 계층은 HTTP 프로토콜과 일반적으로 널리 알려진 HTTP, HTTPS, SMTP 등을 통해 XML 기반의 메시지를 컴퓨터 네트워크 상에서 교환할 수 있는 SOAP(Simple Object Access Protocol) 프로토콜을 이용하여 통신한다.

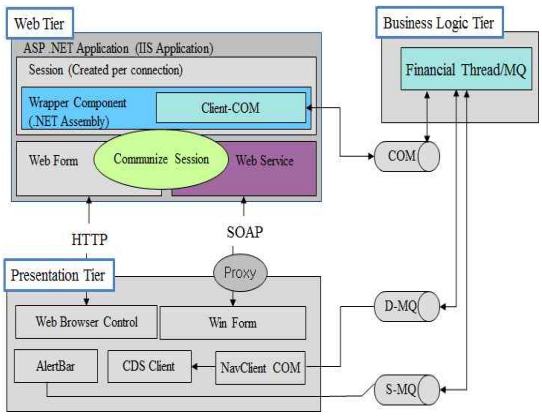

Fig. 1. Communication architecture in each tier.

2.2 주요 성능 개선 활동의 예상 산출물과 요구사항

전산시스템의 성능 개선활동과 그에 따르는 산출물은 일반적으로 시스템 사용자들의 요구사항에 따라 달라진다. 본 논문에서 적용된 금융권 계정계 전산 시스템을 실사용자들에게 오픈하기 위한 준비단계로 시스템의 성능을 파악하기 위한 부하테스트를 실시하였고, 그 결과에 따른 성능 개선 요구사항과 주요 활동 및 예상되는 산출물은 Table 1과 같다.

3. 시스템 성능 개선을 위한 문제 영역 정의

개발 시스템의 주요 요구사항으로 실제 사용자들의 요청을 반드시 처리해야하며, 또한 빠른 응답시간이 요구되는 시스템의 특성을 고려할 때, 가동 후에 발생될 수 있는 다음과 같은 성능카운터와 관련된 코드를 세밀하게 분석할 필요가 있었다.

- 서버 컴포넌트 : Processor Time, Memory Usage, ASP_Net Process Time, ASP Process Time.
- 어플리케이션 컴포넌트 : ASP_Net Process 내에서 활동하는 Code, ASP Work Process내에서 활동하는 Code.

또한, 시스템에서 사용자들의 요구사항을 처리하는 핵심 부분인 데이터 계층과 웹 계층의 성능 개선을 위하여 시스템 구성 요소별 응답시간을 줄일 필요가 있으며, 현재 시스템을 구축하는 은행에서 단말기의 요청처리 속도를 향상시키기 위하여 최종 단말의 응답처리시간을 전체적으로 줄일 필요가 있는 것으로 분석되었다.

추가적으로 비즈니스 로직 계층이나 기타 계층에서 구현된 어플리케이션에 대해서는 작업 프로세스 내에서 활동하는 코드를 분석할 필요가 있으며, 향후

Table 1. The main sector activity and the expected outcomes

Requirement	Activities	Expected Output
[SQL Server] • Request for applying the bad query tuning guide	• Long Run Query List • Find and tune bad queries from the list-based top 30 SQL statement • Strategy establishment for the backup and restore	• Long run & Bad Query List • Quantitative performance improvements list for the before and after tuning • Recovery Report for the DB failure
[Transaction Server] • Request a load test to create a Crash Dump • Request for applying the guide to overcome the Communication Server failure	• Crash Dump Analysis for the communications Server Stabilization • Hang Dump Analysis for the stabilization ClientCOM • MSMQ configuration check • Strategy establishment for the backup and restore	• Analysis report for the communications Server Crash Causes • Analysis report for the ClientCOM hang causes • Optimization guide for the interworking of ClientCOM and Wrapper • Reports for recovering the communication server and the web server failures
[Integrated Terminal] • Request for applying the guide to optimize the memory • Request for applying the guide to optimize the web server and the external processing	• Verify the optimization of the memory usage • verify the web service processing architecture • verify the external processing • MSMQ configuration check	• Guidelines for optimizing memory usage • Guidelines for optimizing the web server • Guidelines for optimizing the external processing • Inspection reports and optimum guide for the MSMQ

발생될 수 있는 다양한 장애를 예방하기 위하여 시스템에서 주 호스트와 다계층 시스템을 구성하고 있는 주요 부분의 서버에 대하여 다양한 유형의 장애를 가정하여 인위적으로 장애를 생성한 후, 이를 분석할 필요가 있었다. 부하를 자동으로 생성하기 위한 부하테스트 툴은 HP사의 부하테스트 소프트웨어인 로드런너(LoadRunner)를 사용하였다.

4. 시스템 성능 분석과 개선

시스템의 문제점 파악하기 위한 자료를 수집하기 위하여 Performance Monitor, DB PSSDiag Tool을 설치한 후, 부하테스트를 수행한 결과 데이터 계층의 DB 서버는 정상적으로 동작하고 있는 반면 비즈니스 로직 계층의 특정 서버가 다운되는 현상이 발생되었으며, ASP.NET Recycling, ODBC Connection Pooling 그리고 DB Lock Timeout과 같은 문제가 발생되는 현상이 관측되었다.

4.1 웹 계층의 성능 개선 사항

우선적으로 웹서버의 Recycling 재발을 방지하기 위한 작업으로 메모리 덤프 분석이 필요하고, 데이터베이스의 성능 분석을 위하여 사용자의 로그인 정보를 저장하는 주요 테이블인 USERDATA과 관련된 인덱스의 효용성을 파악할 필요가 있는 것으로 분석되었다.

웹서버의 재시작 문제는 구체적으로 웹 서버에 과도한 부하가 걸리는 경우에 발생되었으며, ASP.NET이 응답이 없음을 감지한 후, 웹 서버가 자동으로 재시작(Restart)되는 현상이 파악되었다. 이를 방지하기 위하여 Recycling 기능의 사용을 비활성화시키는 방법을 우선적으로 적용하는 방법이 검토되었다.

기존 웹서버는 응답 제한 시간이 3분 이내로 설정되어 있어서, 비정상적으로 동작하는 웹서버에서 설정된 클라이언트의 응답시간 제한시간 동안에 응답이 없는 경우, ASP.NET이 재시작(recycle) 됨에 따라서 모든 세션을 잃어버리는 현상이 발생되었다. 따라서 서버에 접속을 시도했던 모든 클라이언트(Client)에 500 에러가 발생되었다.

이러한 문제를 해결하기 위하여 우측 그림과 같이 “responseDeadlockInterval=Infinite”와 같이 응답

제한 시간을 무한대로 설정을 변경한 후에는 클라이언트(Client)나 서버에서의 응답이 늦어져도 재시작(recycle)이 발생되지 않았다. ASP.NET의 환경설정 파일인 Machine.config에서 해당 파라메타를 값을 설정, 변경하는 예를 Fig. 2에 나타냈다.

설정 변경 후에는, 과도한 부하 시에도 Recycling이 발생되지 않음을 확인하였으며, 그와 관련된 서버 다운 문제도 해결되었다.

```
<processModel
    enable="true"
    timeout="Infinite"
    idleTimeout="Infinite"
    shutdownTimeout="0:00:05"
    requestLimit="Infinite"
    requestQueueLimit="5000"
    restartQueueLimit="10"
    memoryLimit="95"
    webGarden="false"
    cpuMask="0xffffffff"
    userName="SYSTEM"
    password="AutoGenerate"
    logLevel="Errors"
    clientConnectedCheck="0:00:05"
    comAuthenticationLevel="Connect"
    comImpersonationLevel="Impersonate"
    responseDeadlockInterval="Infinite"
    maxWorkerThreads="20"
    maxIoThreads="20"
/>
<webControls
    clientScriptsLocation="/aspnet_client/{0}/{1}/"
/>
```

Fig. 2. An optimization example of setting "Machine.config" file in the ASP.NET.

4.2 비즈니스 로직 계층의 성능 개선 사항

비즈니스 로직 계층에서 셧다운이 발생된 서버(07)와 정상적으로 동작했던 서버들 중에서 20번 서버와 성능카운터의 로그를 비교 분석하였다. 일차 분석결과 윈도우즈 서버의 환경설정 파일인 boot.ini에 설정된 /PAE 옵션과 관련이 있을 것으로 예측되었다. PAE(Physical Addressing Extension) 옵션은 32비트 메모리 어드레싱 영역을 64비트 메모리 어드레싱 영역으로 확장하는 옵션으로, 이를 적용할 경우 시스템이 사용하는 CPU 활용도가 증가될 수 있다. Performance Monitor를 사용하여 동일 시간대 비정상적으로 동작했던 07번 서버와 정상적으로 동작했던 20번 서버에 대하여 Current Connections, Reqests/Sec, % Processor Time의 성능 카운터를 사용하여 수집된 로그로 각각의 관련된 현상을 분석하였다. Fig. 3의 Current Connections 성능카운터 그래프에서 동일 시간대에 07번 서버는 평균 17,478 최대 546,000으로 기록되었으며, 20번 서버는 평균 11,806 최대 542,000으로 Connection 수는 거의 비슷한 것을 볼 수 있다.

Fig. 4에서 Requests/Sec 성능카운터를 사용하여 분석된 07번 서버는 평균 16,048 최대 138,401을 기록했으며, 20번 서버는 평균 15,609 최대 136,333으로 큰 차이가 없는 것을 볼 수 있다.

Fig. 5에서 %Processor Time 성능카운터를 사용하여 동일 시간대를 분석한 그래프에서는 07번 서버는 평균 3,718 최대 32,168을 나타낸 반면, 20번 서버에서는 평균 18,150 최대 91,116번을 기록하여 07번 서버의 CPU 사용량이 훨씬 적은 것으로 분석되었다.

비즈니스 로직 계층에서 일부 서버가 다운되는 현상의 원인을 파악하기 위하여 /PAE 옵션 설정의 영향 분석 결과 서버들이 사용할 수 있는 최소의 Physical 메모리는 4GB이상으로, 분석 대상인 서버들은 이 옵션을 사용할 수 있는 최소 메모리인 4GB

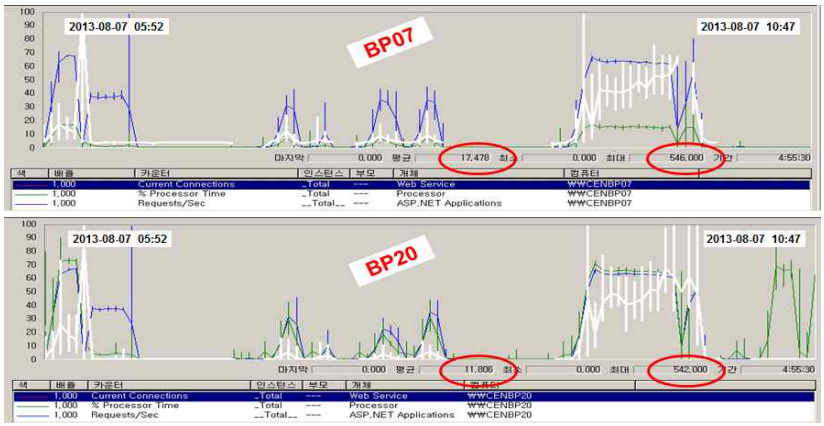

Fig. 3. Analysis the "Current Connections" - 07(top), 20(bottom).

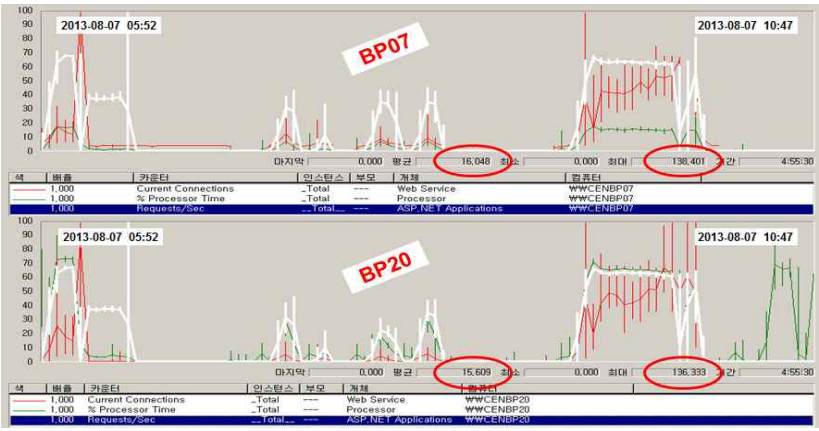

Fig. 4. Analysis the "Requests/Sec" - 07(top), 20(bottom).

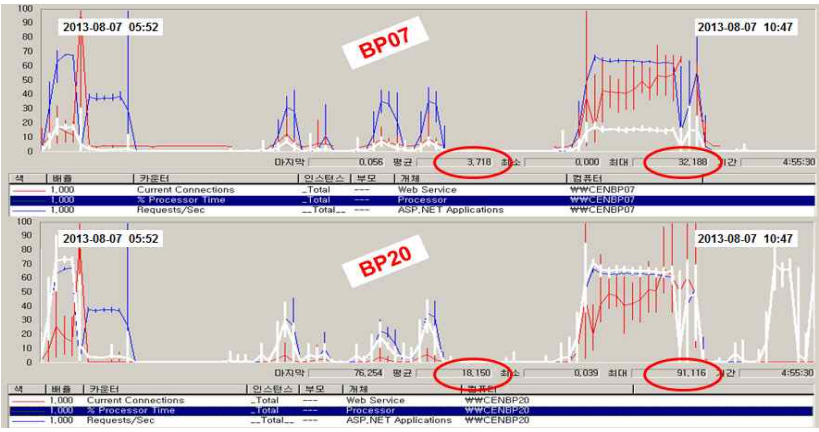

Fig. 5. Analysis the "% Processor Time" - 07(top), 20(bottom).

를 장착하고 있었고, 테스트 시, 정상적으로 동작하고 있는 다른 서버 중에서는 /PAE 옵션이 없는 경우도 있어서 해당 옵션은 성능에 영향을 미치지 않는 것으로 최종 판명되었다.

추가적인 분석에서 5,000명 이상의 사용자가 접근할 경우 2대의 서버가 느려지면서 다운되는 현상은 ODBC Pooling 및 DB Lock Timeout도 그 원인이 아님이 분석되었으며, 보다 정확한 분석을 위해 주요 성능카운터를 사용하여 정상적으로 작동되는 서버와 비정상적으로 작동되는 서버를 비교분석하였다.

Page/Sec 로그로 분석한 가상 메모리 활용률은 정상 작동 서버의 경우 평균 1.4로써 매우 적게 사용을 하였고, 비정상 작동 서버의 경우도 가상 메모리 활용률은 평균 2.8로써 적게 사용을 하였다. 비정상 서버가 다운되는 시점에서는 페이징 횟수 107회의 과도한 Paging이 발생된 것을 볼 수 있었다. 이는 통신 서버다운 시 메모리 사용에 문제가 있었던 것으로 분석되었다. %Processor Time으로 분석한 데이터에서도, 정상 작동되는 서버의 경우 CPU 활용률은 최대 17%, 평균 10%로써 매우 적게 사용을 하였고, 비정상 작동 서버의 경우도 최대 23.4%, 평균 6.4%로써 적게 사용을 하였다. 마찬가지로 서버가 다운된 이후 CPU 사용률이 떨어진 것으로 보아 대부분의 작업들이 CPU를 사용했던 것으로 분석되었다.

4.3 행 덤프를 이용한 서버 간의 통신 문제 개선 사항

일부 서버의 셧다운 현상을 분석하기 정상작동 서버와 비정상작동 서버에 대해 Requests Queued 성능카운터의 로그를 분석하였다. Fig. 6은 Requests Queued 성능카운터로 분석된 로그를 보여준다.

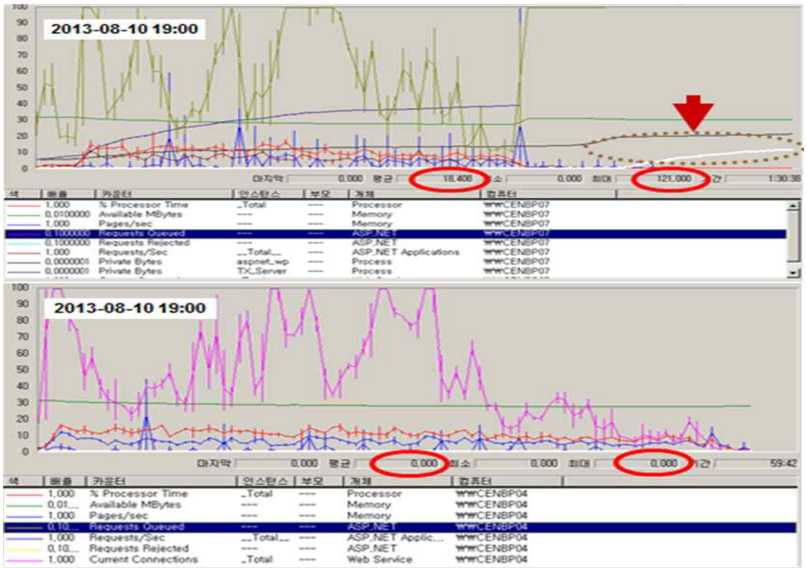

Fig. 6. Analysis the "Requests Queued" - 07(top), 20(bottom).

정상적으로 동작하는 서버의 경우, ASP.NET Queue는 ClientCOM이 서버의 작업 큐에 요청을 전달한 후 작업이 완료 될 때까지 대기하는 요청의 값이 0으로 이것은 Queue에 대기 하는 작업이 없는 것을 뜻하며, 따라서 ClientCOM에 대한 작업 적체 현상은 없는 것으로 판단할 수 있다.

이에 비해 비정상적으로 동작하는 서버의 경우, ASP.NET Queue는 서버가 다운되기 전까지 0이었으나 해당 서버가 다운되면서 ClientCOM의 작업 요청에 대한 처리를 못하므로 ASP.NET의 Queue가 121로 적체되는 현상을 볼 수 있다. 이에 대한 원인을 세밀하게 분석하기 위해 프로세스(Process)가 죽지 않은 상태에서 해당 프로세스에서 사용 중인 메모리의 현황을 그대로 파일로 받아 내리는 행 덤프(Hang Dump)를 사용하여 ASP.NET에서 작업 중이던 프로세스(Work Process)를 분석하였다. ASP.NET 작업 프로세스는 ClientCOM에 대한 쓰레드(Thread)도 함께 포함하고 있는 프로세스로 ASP.NET의 행 덤프를 받는 목적은 병목 현상이 ASP.NET 자체에 있는 것인지 아니면 ClientCOM을 사용하는 프로세스에 있는지 판단하기 위한 검사와 통신 서버로부터의 응답(Response)를 제대로 받고 있는지에 대한 검사를 위함이었다.

덤프 분석 결과, Slow Down이 발생하는 서버는 해당 시점에서 ASP.NET Web Application Process (aspnet_wp.exe)에 147개의 쓰레드(Thread)가 수행 중 이었으며, 147개 중 130개의 쓰레드들은 모두 "clientcom!ClientServiceClass::GetSyncMessage()" 메서드를 호출 한 후, Sync Message를 기다리고 있었는데, 130개 쓰레드 중에서 Critical Section에서의 Lock Contention이 없었다. 따라서 통신 서버의 성능 저하는 clientcom.dll 내의 ClientServiceClass::Get SyncMessage() 메서드에서 병목현상(Bottleneck)이 발생되는 것으로 판단되었으며, 이 메서드에 대하여 통신 서버 쪽의 의존성을 고려하여 성능 개선 작업을 수행하였다. 해당 프로세스를 구현하는 소스코드의 분석결과 프로그램 실행 중에 생성된 주요 객체들에 대한 소멸자 호출을 명시화하는 코드가 구현되지 않았음을 파악할 수 있었고, 문제를 해결하기 위한 코드를 추가하였다. 수정 작업 이후 테스트에서는 정상적으로 동작하는 다른 서버들과 비슷한 CPU 사용률을 보였으며, 메모리 누수도 없어진 것을 알 수 있었고, 해당 서버에서 발생되었던 행(Hang) 현상도 없어졌음이 확인되었다.

4.4 데이터 계층의 성능 개선 사항

데이터 계층에서는 응답시간이 많이 걸리는 쿼리를 분석하기 위해 로그파일로 조사한 결과, 개선이 필요한 3개의 쿼리를 파악하였다. 이 중에서 로그인 처리하기 위해 USERDATA 테이블에 액세스하는

쿼리가 총CPU, 총읽기, 최대실행시간, 최대읽기 등의 항목에서 다른 쿼리들에 비해 월등히 수치가 높은 것으로 나타났으며, 병목현상을 유발하는 쿼리로 분석되었다.

이러한 쿼리에 대해 where 절의 AND 연산과 해당 테이블의 인덱스를 개선하였다. 개선 작업 후, CPU 활용률을 조사한 그래프도 분석해본 결과 로그인 처리에 있어서 쿼리 개선 전에는 SQL Server에서 사용하던 CPU 활용률이 평균 60% 수준이었으나 USERDATA 테이블을 액세스하는 쿼리문에 대한 인덱스를 추가적으로 생성한 후에는 평균 CPU 활용률은 5%대로 낮아졌으며 동시간대에 4,500명이 접근할 수 있었던 것에 비해 8,000명 이상의 로그인 처리가 가능해졌다.

본 사례에서는 다계층으로 구성된 서버에서 비즈니스로직 계층에서 일어나는 특정 서버의 시스템 다운 현상의 원인을 파악하기 위한 첫 번째 단계로 비정상 서버와 정상서버의 주요 성능 카운터의 로그를 비교분석하였는데, 그 중에서 “Requests Queued” 성능 카운터의 로그 그래프가 시스템의 다운 현상이 ASP.Net의 Queue 사용과 관련된 문제임을 대략적으로 보여주었다.

보다 정확한 분석을 위하여 비정상적으로 동작한 서버의 행덤프를 분석하였는데, 서버 다운이 일어나는 시점에서의 행덤프를 분석한 결과 프로그램 실행 중에 생성된 주요 객체들에 대한 소멸자 호출을 명시화하는 코드가 구현되지 않았음을 파악할 수 있었고, 문제를 해결하기 위한 코드를 추가하여 비정상 동작 서버의 시스템 다운 문제를 해결하였다.

5. 결 론

본 논문에서는 실시간으로 고객의 요구사항을 반드시 수행할 필요가 있는 다계층 구조의 금융 전산 시스템에 대하여 실제 사용자들이 사용하기 전에 성능 저하의 원인을 파악하기 위한 방법으로 부하테스트를 수행하는 방법을 보였다.

부하테스트를 수행하기 위한 전단계로 주요 부분별 Activity와 예상 산출물을 정의하였고, 웹 계층의 특정 서버가 리사이클링 되면서 요청된 세션을 읽어버리는 현상에 대하여 성능카운터로 원인을 분석한 다음, ASP.NET의 환경설정 파일의 파라메타 수정을 통하여 서버 다운현상을 해결하였다.

비즈니스 로직 계층에서 발생된 몇 개의 통신 서버가 성능이 저하되면서 결국에는 다운되는 문제점을 개선하기 위하여 성능카운터를 사용하여 원인을 분석하였으며, 구체적인 원인을 파악하기 위해 Page/Sec, %Processor Time, Requests Queued 성능카운터를 사용하여 일차적인 분석을 하였지만 구체적인 원인을 찾아내지 못했고, 서버가 다운되는 시점의 메모리 상태를 파악하기 위해 행 덤프를 받았다. 덤프를 분석한 결과 원인을 제공하는 프로세스를 찾을 수 있었으며, 문제를 일으키는 소스코드 최적화를 통하여 해당 서버의 안정적인 메모리 사용과 행(Hang) 현상을 없앨 수 있었다.

또한, 데이터 계층의 성능 개선을 위해 모니터링 도구가 생성한 로그를 사용하여 성능 개선이 필요한 쿼리를 찾아내고, DBMS에 내장된 쿼리분석기를 사용하여 인덱스를 최적화하는 방법을 보였다.

추후 이러한 연구를 다계층 기반으로 설계되는 다양한 전산시스템에 적용할 수 있도록 일반화시키는 작업이 필요하며, 보다 효율적으로 시스템의 성능을 개선시킬 수 있는 방법에 대한 연구가 필요하다.

REFERENCE

[1] S. Duttagupta and M Nambiar, “Performance Extrapolation using Load Testing Results,” *Proceeding of International Journal of Simulation Systems, Science & Technology*, pp. 66-74, 2008.

[2] T.H.D. Nguyen, B. Adams, Z. Jiang, and A.E. Hassan, “Automatic Load Test Verification using Control Charts,” *Proceedings of the 18th Asia-Pacific Software Engineering Conference*, pp. 282-289, 2008.

[3] D.A. Menascé, “Load Testing of Web Sites,” *IEEE Internet Computing*, pp. 70-74, July·August 2002. http://computer.org/internet/

[4] S.W. Lee, J.S. Kim, and T.S. Kim, “Optimization of DB Server and Web Server to Enhance the Performance of ECM,” *Journal of Korea Multimedia Society*, Vol. 16, No. 12, pp. 1446-1453, 2013.

[5] J.W. Kim, "Bandwidth Requirement and Priority-based Synchronization Methods in Hybrid Client-Server Architecture for Mobile Multiplayer Games," *Journal of Korea Multimedia Society*, Vol. 17, No. 4, pp. 526-534, 2014.

[6] J. Park, J. Kim, and T. Kim, "Analyzing Operation Deviation in the Deasphalting Process using Multivariate Statistics Analysis Method," *Journal of Korea Multimedia Society*, Vol. 17, No. 7, pp. 858-865, 2014.

[7] Windows Sysinternals, http://www.microsoft.com/technet/sysinternals (accessed Apr., 20, 2015).

[8] PSSDIAG Data Collection Utility, http://support. microsoft.com/default.aspx?scid=kb;en-us;830232 (accessed Feb., 14, 2012).

[9] Debugging Tools for Windows(WinDbg, KD, CDB, NTSD), http://msdn.microsoft.com/ko-kr/library/windows/hardware/ff551063(v=vs.85).aspx (accessed Feb., 12. 2014).

[10] SQL Server Disk Performance Metrics - Part 2 - other Important Disk Performance Measures, http://www.sqlshack.com/sql-server-disk-performance-metrics-part-2-important-disk-performance-measures/ (accessed May., 12, 2014).

[11] How To: Use SQL Profiler, https://msdn.microsoft.com/en-us/library/ff650699.aspx (accessed Feb., 12, 2014).

[12] Microsoft TechNet-Evaluating Memory and Cache Usage, https://technet.microsoft.com/en-us/library/cc958290.aspx (accessed Jan., 12, 2014).

[13] Microsoft TechNet, Library, Server Object, https://technet.microsoft.com/en-us/library/cc757730(v=ws.10).aspx (accessed Jan., 12, 2004).

[14] MSDN Library - Memory Object, https://msdn.microsoft.com/en-us/library/ms804008.aspx (accessed Mar., 08, 2015).

김 태 석

1981년 경북대학교 전자공학과 공학사 졸업
1989년 일본 KEIO대학 이공학부 계산기과학전공 공학석사
1993년 일본 KEIO대학 이공학부 계산기과학전공(공학 박사)
1993년 일본 국제전신전화연구소(KDD) 기술고문
1993년 일본 KEIO대학 이공학부 객원연구원
1994년~현재 동의대학교 컴퓨터소프트웨어공학과 교수
관심분야 : 정보시스템, 기계번역, 인터넷비즈니스

이 종 윤

1994년 명지대학교 산업공학사 졸업
1996년 고려대학교 산업공학석사
2015년 동의대학교 컴퓨터소프트웨어공학전공 공학박사
2006년~현재 한국마이크로소프트 금융사업본부 기술영업대표
관심분야 : 데이터베이스, 소프트웨어 설계

김 종 수

1992년 부경대학교 냉동공학 전공 공학사 졸업
2003년 부산외국어대학교 컴퓨터공학전공 공학석사
2006년 동의대학교 컴퓨터소프트웨어공학전공 공학박사
2012년 동의대학교 산업기술 개발연구소 P.D 연구원
2013년~현재 한국승강기대학 승강기공학부 교수
관심분야 : 소프트웨어 설계, 게임