논문 13-38B-12-06

한국통신학회논문지(J-KICS) '13-12 Vol.38B No.12
http://dx.doi.org/10.7840/kics.2013.38B.12.976

# 분산네트워크 환경에서의 Logging 기반 BTM 설계 및 구현

김 용 옥◆, 최 용 락*, 성 기 범**, 이 남 용***, 김 종 배°

# Logging Based System Design and Implementation of Business Transaction Management for a Distributed Network Environment

Yong-Ok Kim◆, Yong-Lak Choi*, Ki-Bum Sung**, Nam-yong Lee***, Jong-Bae Kim°

요 약

전통적인 IT 관리 솔루션들의 주요 관리 대상은 시스템과 네트워크였다. SMS, NMS 등의 솔루션들이 관리 솔루션의 대명사였던 시기가 있었지만 오늘날 하드웨어 발전 속도가 빨라지면서 성능향상의 관점은 자연스럽게 하드웨어에서 소프트웨어로 바뀌어 가고 있는 추세다. 즉, 효율적이고 최적화된 애플리케이션의 성능 관리가 비즈니스의 성패를 좌우한다고 해도 과언이 아닌 상황이 된 것이다. 애플리케이션 성능관리를 위해 다양한 형태의 시도가 이루어지고 있는데 본 연구에서는 실시간 애플리케이션의 운영 현황을 모니터링하고 신속한 장애대응을 가능하게 하기 위한 방법으로 컴포넌트(Component)간의 호출 및 응답에 대한 상관관계를 가장 효율적으로 가시화하고 능동적인 성능관리가 가능한 방법으로 API에 의한 로깅시스템(Logging System)을 제시한다.

**Key Words** : Transaction, performance, risk, Realtime, APM

ABSTRACT

The targets for the traditional IT management solutions has been mostly systems (server) and networks. Such that management system for system and network, SMS and NMS respectively, used be the IT management solutions. Now the speed of hardware improvement is faster than ever, the performance improvement scope is shifting from hardware towards software. Therefore, it is not arguable that the business success depends on the efficient and optimized application performance management. There are many different approaches are developed in the application performance management, but in this study, API-based logging system that most efficiently visualizes the correlation of call and response between Components and enables active performance management for monitoring real-time applications operation status and resolving the problems and failures quickly.

## Ⅰ. 서 론

웹과 CBD(Component Based Development) 개발 방법론에 의해 개발되는 오늘날의 애플리케이션들은 과거 C언어 등의 절차적 처리 방식에 비해 복잡하고 그 관리 포인트도 다양하기 때문에, 장애가 발생하는 경우 그 원인을 찾아내기가 쉽지 않고 심지어는 장애지점을 확인하는 것조차도 어려운 상황

◆ First Author : 유피니트, Yong-Ok Kim, 정회원
° Corresponding Author : 숭실대학교 SW특성화대학원, kjb123@ssu.ac.kr, 정회원
* 숭실대학교 SW특성화대학원, ylchoi58@ssu.ac.kr, **대검찰청, sung1177@daum.net, 정회원
***숭실대학교 컴퓨터학부, nylee@ssu.ac.kr, 정회원
논문번호 : KICS2013-10-444, 접수일자 : 2013년 10월 14일, 심사일자 : 2013년 11월 4일, 최종논문접수일자 : 2013년 11월 22일

이다. 이런 상황에 대처하기 위한 많은 노력들이 이루어지고 있는데, 애플리케이션 전 구간에 걸친 거래의 추적과 모니터링을 통하여 각 구간의 성능을 감시하고 각 애플리케이션 Tier별 응답시간을 관리하려는 시도가 그것이다.

시스템 아키텍처의 변화 측면에서 엔터프라이즈 시스템은 메인프레임, 오픈시스템, 모바일 및 클라우드 시스템으로 급격하게 변하고 있으며, 그 무게중심이 운영 관리를 위한 시스템 중심에서 상품 개발을 위한 비즈니스 중심으로, 다시 사용자 UX 중심으로 중심축이 빠르게 이동하고 있다.

이러한 시스템 관리관점의 변화에 따라 IT 시스템 관련 솔루션 벤더들은 최종사용자 관점에서의 서비스 품질향상을 위해 다양한 솔루션들을 내 놓고 있는 상황이며, 그 방향은 사용자 체감속도에 기반한 성능(Performance) 관리, 애플리케이션 자체의 성능 및 장애 관리 등으로 변화해 가고 있다[1,2].

애플리케이션 라이프 사이클 프로세스를 보면 IT 시스템 기획 및 투자, 분석/설계, 개발, 이관, 운영/유지, 소멸의 단계로 이루어지는데 이때 운영/유지 단계가 가장 핵심이고 전체 프로젝트의 궁극적인 목적이 되는 부분이기에 운영단계에서의 유지보수와 사용자 만족도를 높이는 것이 무엇보다 중요하다.

이런 IT 환경의 변화와 필요에 의해 최근 몇 년 사이 웹 애플리케이션의 성능관리를 목적으로 다양한 솔루션들이 APM(Application Performance Management)[3] 이라는 새로운 영역으로 발표되었고, 시장에서 어느 정도 성숙한 솔루션으로 인정받고 있으나 웹과 Legacy 시스템이 혼재하는 복잡하고 대규모의 시스템관리를 위해서는 새로운 형태의 기술을 적용한 솔루션이 필요하다.

본 연구에서는 애플리케이션 전 구간에서 거래를 투명하게 관리하고 이를 기반으로 장애 혹은 성능저하가 발생 할 경우, 그 정확한 원인 구간과 원인이 되는 애플리케이션을 신속하게 찾아냄으로써 빠른 조치가 가능하고 나아가 지속적인 모니터링을 통하여 튜닝 대상이 되는 애플리케이션을 찾아내어 성능을 향상시킬 수 있는 관리용 솔루션을 설계하고자 한다.

## Ⅱ. 관련연구

기존의 정보시스템 성능 관련연구는 장비도입시의 규모산정을 위한 성능측정기준에 관한 연구와 네트워크 등 물리적 장비의 장애 유무, 단순 사용량 측정 등 장비관리 기능과 장애관리업무에 관한 연구[4-8]를 주로 수행하였다.

최근 많은 솔루션들이 애플리케이션 관점에서의 성능관리 솔루션들을 발표하고 있는데[2], 비즈니스는 사용자가 사용하는 애플리케이션을 통해서 발생하는 것이므로 진정한 성능관리란 클라이언트-네트워크-서버간 애플리케이션 라이프 사이클 전 과정을 통해서 측정하고 관리하는 애플리케이션 성능관리(APM)를 의미한다[3].

오늘날의 애플리케이션들은 웹을 포함하여 애플리케이션 Tier를 별도로 구분하고 있으며 거기에 EAI 솔루션을 이용하여 외부 기관 연계 등과 같이 타 업무 시스템과의 연계를 위하여 또 하나의 Tier가 존재하고 이러한 다단계의 Tier들이 관제대상이 되고 있다. 즉, 하나의 애플리케이션이라 할지라도 한 시스템에서 동작하는 것이 아니고 분산 아키텍처에 의해 여러 시스템에 걸쳐서 실행되기 때문에 이를 관제하기 위해서는 전 시스템의 거래현황을 시스템 단위가 아닌 애플리케이션 거래중심으로, 즉 비즈니스 트랜잭션(Business Transaction) 단위로 관리하고 통제할 수 있는 솔루션을 필요로 한다.

## Ⅲ. 거래추적 시스템 설계

### 3.1. 거래의 정의

금융시스템에서의 거래란 텔러에 의해 최초에 발생되며 특정업무 처리를 위해 Web을 통해 접속하여 정보계 서버를 통하여 계좌 정보를 획득하고, 이를 기반으로 다시 계정계 시스템에 접속하여 해당 거래를 완료하는 것을 말하며, 계좌 이체 처리 시에는 타 기관의 정보계와 계정계 시스템까지 연계되는 일련의 행위를 말한다.

일반적으로 계정계 시스템은 오랜 기간 BANCS[8] 라는 코어 뱅킹솔루션 패키지를 사용하며 언어는 COBOL이나 C이지만 정보계 시스템이나 타 단위 시스템들은 JAVA 애플리케이션으로 개발하고 있는 상황이기 때문에, 하나의 거래는 웹 애플리케이션과 티피모니터 기반의 애플리케이션으로 되어 있다.

### 3.2. 거래 구분을 위한 Unique Code ID

일반적인 금융 애플리케이션들은 거래 전문을 활용 하고 있는데 각 구간의 거래를 하나의 트랜잭션으로 보고 관리 포인트를 통일하기 위한 Unique ID로 GTRID(Global Transaction ID) 라는 개념의 관

리용 코드를 거래 전문에 삽입하는 방식을 사용하고자하며, 효과적인 관리를 목적으로 GTRID는 다음과 같이 구성 하도록 한다. 물론 상황에 따라 각종 정보성 항목은 변경 될 수 있다.

표 1. 필수 Logging 항목
Table 1. Required logging entries

| Type | Name | Length | Description |
|---|---|---|---|
| | E2E Log Start Delimiter | 1 | 0x02 (ASCII Code - STX) |
| | Log Flag | 1 | * (E2E Start / End / Error / Information Log) |
| Header | Log Type | 1 | S(Start), F(End), E(Error), I(Information) |
| | GID | 30 | UNIQUE ID |
| | SEQ | 3 | 900, 910, 920 |
| | Transaction Code | VAR | JOB ID ([Code][Code][Y][Job Code]_[0][0][0][0]) ex) LTYB_0001 |
| | Logging Date | 17 | Current Date(YYYYMMDD) + Current Time(HHMMSSMILLI) |
| | Processing Time | VAR | E2E Automatic Create From Agent. |
| | System Name | VAR | "hostname(hostip)" (system ID) |
| | Service Name | VAR | Class Name (ex. hi.common.batch.test.ZZYT0001.step01.ZZYT000101BT) |
| | Group ID | VAR | Group ID (ex. ZZYT000101GRP) |
| | Step ID | VAR | Step ID (ex. ZZYT000101BT) |
| | Channel ID | VAR | Channel ID (OLTP,CTL+M, ...) (Start Position Information) |
| | Scheduled ? Non Scheduled | VAR | 1(Scheduled), 2(Non Scheduled) |
| | RETRY_GID | VAR | Before GID when Restart case (No Information, Non Restart Case) |
| Info | Job Config or Info_Msg | VAR | JOB Configuration Information ("SEQ" == 900 && "Log Type" = S Case) Job Configuration , else : info Message |
| Error | Error Msg | VAR | Error Message |
| Trace | Trace Msg | VAR | Trace Message |
| | E2E End Log Type | 1 | 0x03 (ASCII Code - ETX) |

Ex Normal ) [STX]*[GS]S,001A6B66A3332007091110203615 60,900,ZZYT0001,20070911101914483,,FrameworkMA(app-was1/10.52.1.71:7101),com.apptomo.batch.test.ZZYT0001.step01.ZZYT000101BT,ZZYT000101GRP,ZZYT000101BT,channel#1,1,,[GS]JOB_Conf or info_msg[GS]error_msg[GS]trace_info[GS][ETX]

Ex ReStart) [STX]*[GS]S,001A6B66A3332007091110203615 60,900,ZZYT0001,20070911101914483,,FrameworkMA(app-was1/10.52.1.71:7101),com.apptomo.batch.test.ZZYT0001.step01.ZZYT000101BT,ZZYT000101GRP,ZZYT000101BT,channel#1,1,101A6B66A3332007091110203615 61,[GS]JOB_Conf or info_msg[GS]error_msg[GS]trace_info[GS][ETX]

표 1 및 표 2와 같은 거래 로그를 남기기 위하여 해당 각 거래 시작과 종료 시점에 공유메모리에 로깅 할 수 있는 API를 제공하여 프레임워크에서 이를 처리 하도록 하여 필요한 로그들을 수집하고, 이 과정을 통하여 수집된 거래로그들은 에이전트와 거래추적 서버에서 각각 거래추적에 활용 된다.

### 3.3. 거래추적 에이전트 설계(Agent Design)

업무시스템 개발시점에서 공통모듈 혹은 프레임웍 단에서 표 1 및 표 2와 같은 항목을 모든 서비스 호출 전후에 로깅하고 거래추적 에이전트는 이 로그를 거래추적 서버에 전송하는 구조인데, 로깅 메커니즘은 Gateway, Main Service, Sub Service, DBIO Adapter, Rule Adapter, Internal Adapter, External Adapter 등의 구간에서 시작/종료 거래 로그를 남기고 이를 기반으로 거래 간 연계 맵을 완성하는 것을 기본으로 한다.

거래추적 에이전트는 애플리케이션 서버에 탑재되는 프로세스 형태이기 때문에 운영시스템에 부하를 최소화 하여야 한다는 대 전제를 가지고 있다. 따라서 에이전트의 역할을 최소화 할 수 있도록 설계되어야 하기 때문에 그 기능 역시 최소화 하였다. 일반적으로 은행 뱅킹시스템의 처리량은 2000 TPS - 2500 TPS 수준으로 알려져 있는데, 이 정도 규모의 시스템을 예상한다면 초당 150,000건(약 80M 바이트)의 로그처리가 가능해야하고 하루 평균

표 2. 로그 구성
Table 2. Log Configuration

| Type | Format | Description |
|---|---|---|
| Normal Log | S | Normal Start Log |
| | P | Process Stage Log |
| | F | Normal End Log |
| | G | Normal End Log Without Start Log |
| Error Log | O | Timeout Log |
| | E | Error End Log |
| | D | Error End Log without Start Log |
| Infor-mation Log | I | Information Log |
| | | Debug Log |
| | | Error Log |
| | | Trace Log |
| | | Framework Log |

500GB의 데이터 처리가 가능해야 한다. 이 점을 고려하여 로깅 API가 남기는 모든 거래 로그는 공유메모리 큐의 형태로 남기고 클라이언트 에이전트는 이 로그를 서버에 전송하는 역할만으로 그 기능을 한정 하였다.

다만 공유메모리에 로깅하기 전에 응답시간을 체

크하여 이미 설정된 임계치를 초과하는 거래에 대해서는 거래추적 서버의 별도 포트를 이용하여 데이터를 전달함으로써 실시간 모니터링에 의한 응답시간 관리가 가능하도록 설계 하였다.

동시에 SMS나 메일 등을 이용하여 해당업무 담당자에게 Notify 하는 기능을 구현 하게 된다면 업무 협업의 관점에서 많은 도움이 되겠지만 어디까지나 부가 기능의 일부이기에 본 연구에서는 다루지 않도록 하겠다.

### 3.4. 거래추적 시스템 설계

각각의 애플리케이션 서버와 에이전트는 1:1로 탑재가 되지만 에이전트와 거래추적 서버의 관계는 "1 : N" 형태로 설계되어야 한다. 그만큼 대량 데이터를 처리하기 위한 서버의 용량도 확보 되어야 한다는 것이다.

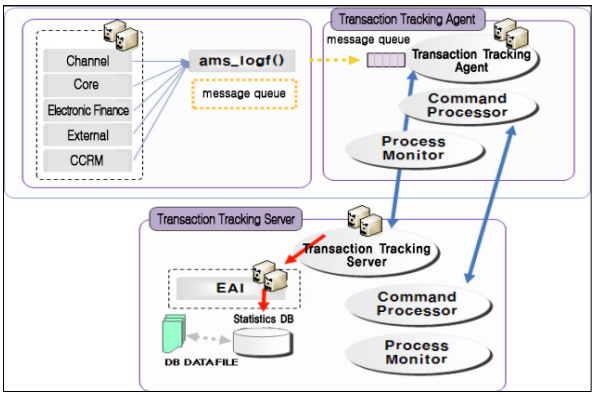


그림 1. Agent & Server 구조
Fig. 1. Agent & Server architecture

에이전트와 서버의 구조는 그림 1과 같다. 에이전트는 거래정보, 시스템 자원 체크, 모든 거래의 임계치 관리 등의 실시간 처리를 주로 담당하며 거래추적 서버와는 NIO 통신을 하도록 설계하여 대량 데이터 처리에 영향이 없도록 하였다.

공유 메모리상에서 이루어지는 처리 과정을 도식화 하면 그림 2와 같다.

#### 3.4.1. 거래추적 에이전트

거래추적 에이전트의 가장 중요한 요소는 운영중인 애플리케이션에 최대한 영향이 없도록 하여야 한다는 것 이다. 시스템에 미치는 부하(Overhead)를 최소화 하기 위하여 에이전트 자체의 기능 또한 최소화 하고 최적화 하는 것은 무엇보다 중요한 요소이다.

거래추적을 위하여 애플리케이션 공통모듈에서는

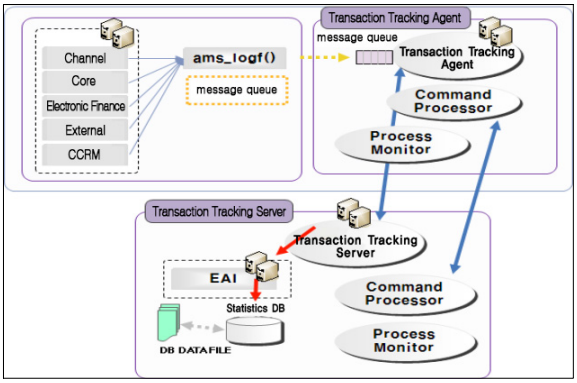


그림 2. 거래 추적 프로세스간 연계
Fig. 2. Linkage between business transaction process

거래를 시작하기 전/후에 각각의 시간을 기록(Time Stamp)하는데 이때 표 1과 같은 항목으로 로깅 메세지를 완성하여 메모리나 파일 시스템에 로깅 하기 때문에 모든 거래는 최소한 한 쌍(시작로그와 종료로그)의 거래 로그를 남기고 에러 상황이나 이상

거래 발생 시 추가적으로 관련 로그를 표 2에 따라 로그를 분류하여 남기는 시스템이다.

거래추적 에이전트의 가장 기본적인 역할은 이렇게 남겨진 로그를 거래추적 서버에 전달하는 것 이다. 대형 시스템의 경우 대량의 로그가 발생하기 때문에 이를 처리 하는 과정에서 디스크 I/O나 네트워크에 부하를 유발하여 자칫 운영서버에 영향을 줄 수 있으므로 전송할 데이터의 양을 고려하여 전송 주기나 전송 사이즈를 결정 하여야 한다.

#### 3.4.2. 거래추적 서버

거래추적 서버에서는 그림 2와 같이 크게 3가지 프로세스가 동작을 하는데 기본적으로는 거래추적 에이전트와의 통신을 통하여 각 에이전트들이 수집하는 성능 관련 데이터를 수집하는 일을 한다.

또한 거래추적 서버와 애플리케이션 서버의 에이전트 프로세서 자체를 모니터링 하는 프로세스와 거래추적 서버로부터 해당 AP 서버의 거래추적 기능을 On/Off 시키는 등의 명령을 받아 수행하는 코멘드(Command) 처리 도구를 에이전트 기능의 일부로 추가함으로써 에이전트의 관리와 프로세스 정상동작 여부를 모니터링 하도록 하였다.

또한 거래 시 발생하는 모든 로그들을 효율적으로 관리하기 위하여 표 1과 같은 일자별 테이블을 생성하고 또 일자별 데이터의 통계를 별도로 관리하는 통계테이블을 생성하였다. 일자별 테이블은 1주일 간격으로 초기화하여 디스크 활용을 최소화

하고 히스토리 관리는 통계테이블을 활용할 수 있도록 하였다.

필요한 경우 데이터 압축이나 암호화 등의 기능은 별도로 설계하여 적용한다면 보다 효율적이고 보안에 강력하게 대응 할 수 있겠지만 이번 연구 대상에서는 제외 하였다.

#### 3.4.3. API 모듈

개발 공통 모듈에서는 일관성있는 로그를 생성하기 위하여 ams_logf()라는 함수를 이용하는데 이 함수를 이용하여 거래를 시작하고 종료시점에 이 함수를 호출 하는 것만으로 표 3과 같이 로그정보를 자동으로 생성하여 파일시스템에 로깅을 할 수 있도록 자동화 하였다. 표 3은 시스템 시간을 자동 추출하여 파일에 기록하는 예제 소스이다. 이와 같이 로깅 항목에 필요한 각각의 항목들은 자동으로 추출하고 로그 구분 플래그, 프로그램 명 등의 자동 수집이 불가한 항목은 ams_logf()함수 호출 시에 파라메타 형태로 기록하도록 작성하여 개발자의 수작업을 최소화 하도록 하였다.

표 3. 자동화된 Logging 예
Table 3. Automated Logging Example

```
int ams_logf(char *sFormat)
{
    /*  Example for Logging function,    */
    /* Variable Declaration */
    va_list ap;
    time_t now_time;
    struct tm *ct;
    char log_file[256];
    char  sTemp[2048], sTempBuf[256];
    FILE *fd;

    va_start(ap, sFormat);
    /* Get System Time for Time stamp */
    time(&now_time);
    ct = localtime(&now_time);
    memset(log_file, 0x00, sizeof(log_file));
    /* Modify System Time To Log Format */
    sprintf(log_file,"%s%04d%02d%02d.log",
          log_path, ct->tm_year+1900,
            ct->tm_mon+1,
            ct->tm_mday);
    fd = fopen(log_file,"a");
```

### 3.5 거래추적 시스템 콘솔

거래추적 Agent로 부터 전송 받은 거래정보들은 거래추적 서버에서 분석되어 장애 트랜잭션 분석, Long 트랜잭션 분석, 실시간 성능 모니터링, 애플리케이션 구간별 성능 분석, 트랜잭션 평균 처리속도 분석, 애플리케이션 튜닝 포인트와 같은 정보를 거래추적 콘솔을 통해 관리자에 제공함으로써 성능 저하 구간을 직관적으로 판단할 수 있게 된다.

에이전트를 이용하여 수집하는 로그에는 업무코드, 서비스 시작시간, 서비스 종료시간, 서버명 등의 정보가 GTRID 기준으로 수집되기 때문에 동일한 GTRID를 서비스 시작시간을 기준으로 정렬하여 해당 트리를 구성하면 거래에 대한 상세 정보를 도식화 할 수 있고 또한 다양한 형태의 통계자료를 바탕으로 각종 분석 자료를 제공할 수 있게 된다.

## Ⅳ. 적용 사례 및 효과

### 4.1. 적용 내역

본 거래추적 시스템을 메이저 화재보험사의 시스템개발 시점에서 업무 오픈 시까지 지속적으로 적용 하였다. 먼저 해당 프로젝트의 표준화 팀과 협의를 통하여 GTRID와 로깅 항목을 선정하고 프레임웍에서 모든 거래를 호출하기 전과 결과를 리턴하기 전에 해당 로그를 남기도록 하였다.

로그 수집은 8 Core CPU와 24 Giga 메모리를 장착한 서버를 이용하였고 데이터베이스는 오라클을 사용하였으며 거래추적을 위한 서버의 이중화는 구성하지 않았다.

거래추적 대상 애플리케이션은 웹 단에서는 웹로직 WAS 기반이고 AP구간에서는 TP모니터인 TMAX 미들웨어를 사용하는 총 8대의 AP 서버에 거래추적 에이전트를 설치하였으며 부하 발생기를 이용하여 전체 150 User를 이용하여 900 TPS를 5분간 총 270,000 트랜잭션(Transaction)을 발생시켜 부하 테스트를 진행 하였다.

### 4.2. 적용 결과

전 거래의 응답 시간을 관리함으로써 얻을 수 있는 중요한 정보 중의 하나는 응답시간, 거래량의 추이를 확인할 수 있다는 것이었다. Peak Day와 비교한 거래량 추이, 어제 등 과거 특정일과의 거래량, 응답 시간 추이를 확인함으로써 전체적인 처리 현황을 파악 할 수 있기 때문이다.

또한 응답이 불량한 거래리스트, 즉 Top N 트랜잭션 리스트를 지속적으로 개발팀에 제공함으로써

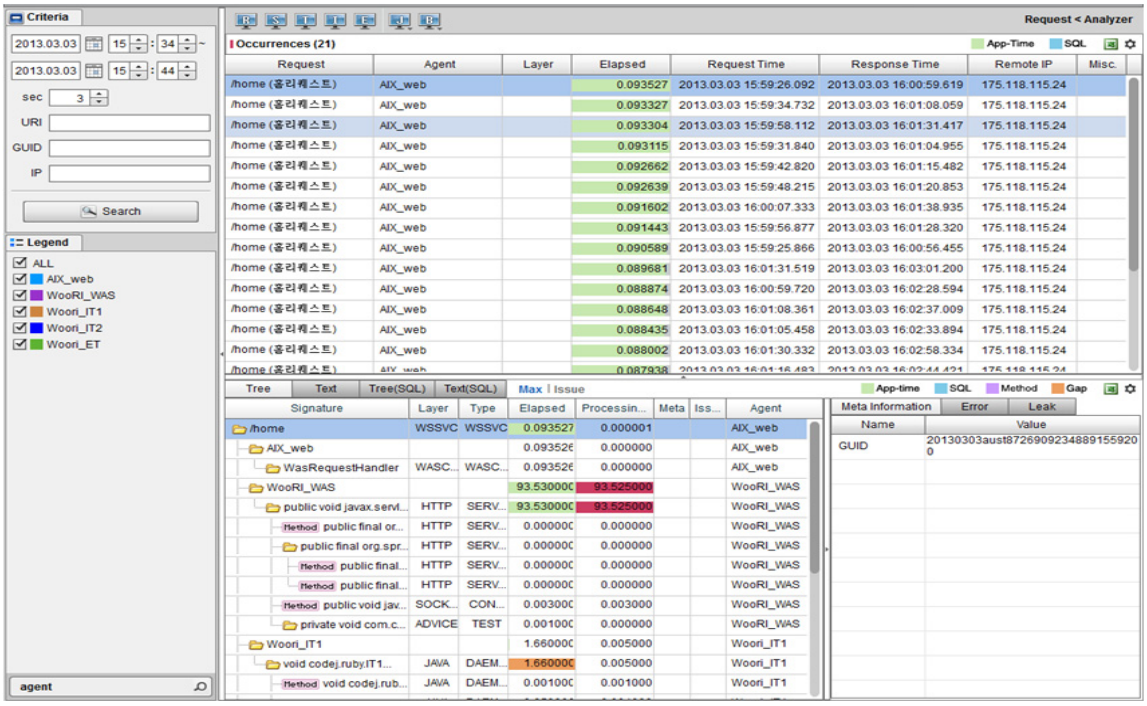


그림 3. 응답시간 별 Top 20 화면
Fig. 3. Top 20 screen by response time

성능 개선을 유도 하였고 이후에는 또 다른 Top N 리스트를 반복적으로 제공하여 지속적인 성능 개선 효과를 가져 올 수 있었다. M 화재에서는 그림 3과 같이 Top 20리스트를 이용하여 부하테스트와 성능개선 작업을 병행하였다.

표 4에서 보는 바와 같이 동일한 조건으로 부하테스트를 수행하고 그 결과를 바탕으로 Top 10 거래 튜닝을 실시하는 방법으로 4회에 걸쳐 테스트를 반복 한 결과, 평균 응답시간 기준 35%의 성능 향상을 실현 하였다. 완벽한 튜닝이 되지 않은 시스템을 대상으로 한 테스트였기 때문에 개선율 자체는 다소 높은 편 이었지만 분명한 포인트를 잡을 수 있었고 또한 장애 발생 시에도 정확한 발생 구간을 손쉽게 파악하여 조치 할 수 있었다.

또 다른 측면에서는 응답시간에 대한 SLA를 적용하여 임계치를 초과하는 거래가 발생하는 경우 관리자가 즉시 이를 확인 할 수 있었고, 이 경우 웹 구간인지 미들웨어 구간인지를 손쉽게 파악함으로써 관리자로 하여금 조치 포인트를 찾아가는데 많은 도움이 되었다.

또한 통계 리포트를 통하여 서버별 업무처리량을 파악할 수 있게 하여 일정기간 거래추적 시스템을 적용하여 운영하게 되면 이 통계를 바탕으로 업무량을 기반으로 한 애플리케이션을 보다 효율적으로 분산 배치하거나 물리적으로 하드웨어 리소스가 부족하다고 판단 할 경우 서버 증설 등의 근거 자료로 활용 할 수도 있을 뿐만 아니라 정확한 증설 대상을 선정하는 데에도 객관적인 자료로 활용 될 수 있음을 확인 하였다.

표 4. M화재 성능 테스트 결과
Table 4. Results of M Fire Insurance Performance Test

| Type | 1st | 2nd | 3rd | 4th |
|---|---|---|---|---|
| User | 500 | 500 | 500 | 500 |
| TPS | 900 | 900 | 900 | 900 |
| Total Count | 27 | 27 | 27 | 27 |
| Average Res. Time | 3.874 | 3.064 | 2.995 | 2.506 |
| Max Res. Time | 8.543 | 6.106 | 5.07 | 4.009 |
| Min Res. Time | 1.087 | 1.099 | 1.077 | 1.079 |
| Error Count | 67 | 54 | 45 | 34 |

## V. 결 론

현업에서 많은 애플리케이션 관리자들을 만나보면서 느끼는 점은 대부분 일관적 이었다. 과거 메인프레임 시절의 애플리케이션 관리 방식은 다운사이징이 되고 턱시도, 티맥스 등의 미들웨어기반의 C나 COBOL 프로그래밍 되면서도 크게 달라지지 않

았다는 것이다.

그러나 프로그램 환경이 웹으로 변화되고 EAI, MCI 등의 솔루션으로 이기종 애플리케이션 통합이 되면서 그 관리 포인트도 다양해지고 하나의 거래가 하나의 서버 내에서 처리되지 않고 복수의 서버에서 이루어지다 보니 애플리케이션 관리가 복잡할 뿐 아니라 장애가 발생 하는 경우에도 그 정확한 지점을 찾아내기가 어렵다는 것이다.

이번 연구의 주된 과제는 이런 관리자들의 고충을 덜어주기 위한 방편으로 애플리케이션과 트랜잭션의 가시성을 제공하고 그 가시성을 기반으로 시스템 운영상황을 애플리케이션 측면에서 보여줌으로써 SMS(System Management Solution) 기반 하 에서의 시스템관리에 의존하던 기존의 관리방식에 AMS(Application Management Solution)라는 새로운 관리 모델을 제공 하고자 하였다.

전 거래를 감시하고 업무 중심의 거래 관리를 함으로써 현업과의 Gap을 줄이고 응답시간에 근거한 지속적인 애플리케이션 튜닝을 지원할 수 있는 도구로 활용된다면 시스템 관리자, 애플리케이션 관리자 그리고 최종 사용자에 이르기까지 효과를 체감할 수 있을 것이다.

또한 이러한 지속적인 활동은 궁극적으로 서비스의 향상을 가져올 것으로 기대한다.

## References


[1] G. Khanna, K. Beaty, G. Kar, and A. Kochut, "Application performance management in virtualized server environments," in *Proc. IEEE/IFIP Network Operations Management Symp.*, pp. 373-381, Vancouver, Canada, Apr. 2006.

[2] H. W. Kim, J. S. Han, N. Y. Lee, and J. B. Kim, "A study on SLA criteria for performance management based APM," *J. Korea Inform. Commun. Soc. (KICS)*, vol. 35, no. 12, pp. 257-262, Dec. 2010.

[3] S. K. Park and D. J. Choi, "Response time-based web service availability measurement method," *J. Korea Inform. Process. Soc. (KIPS)*, vol. 10, no. 1, pp. 61-70, Feb. 2003.

[4] J. H. Na and K. D. Choi, "An exploratory study on capacity sizing method for information system: focus on H/W sizing in pubic sector," *J. Korea Soc. Syst. Integration*, vol. 3, no. 2, pp. 9-23, Nov. 2004.

[5] H. Y. Joung. J. H. Na, and K. D. Choi, "An empirical study on hardware capacity sizing for information systems," *J. Nat. Inform. Soc. Agency*, vol. 12, no. 3, pp. 54-72, Feb. 2005.

[6] M. S. Lee and C. H. Park, "Design and implementation of a web-based traffic monitoring and analysis system," *J. Korea Inst. Inform. Sci. Eng. (KIISE)*, vol. 29, no. 6, pp. 613-624, Dec. 2002.

[7] J. Cleary, I. Graham, T. McGregor, xS. Donnylly, J. Curtis, L. Ziedins, M. Pearson, J. Martens, and S. Martin, "High precision traffic measurement," *IEEE Commun. Mag.*, Vol. 40, No. 3, pp. 167-173, Mar. 2002.

[8] Curocom, BANCS, retrieved Jan. 17. 2013, from http://www.curocom.com.



**김 용 옥** (Yong-Ok Kim)

2010년~현재 숭실대학교 대학원 IT정책경영학과 박사과정

2004년~현재 (주)유피니트 대표이사

<관심분야> 어플리케이션성능관리, SLA, ITSM

**최 용 락** (Yong-Lak Choi)

2002년 8월 숭실대 대학원 박사

2002년~2005년 세종대학교 교수

2006년~2011년 숭실대 정보과학대학원 겸임교수

2012년~현재 숭실대 소프트웨어특성화대학원 교수

2013년~현재 (사)한국정부회계학회 회장

<관심분야> 데이터베이스, 데이터 모델링, 빅 데이터, 소프트웨어 공학, 오픈소스소프트웨어

**성 기 범** (Ki-Bum Sung)

2003년 2월 숭실대학교 컴퓨터학과 석사
2012년~현재 숭실대학교 IT정책학과 박사과정
2011년~현재 대검찰청 DFC 사무관
<관심분야> DB, 디지털포렌식, IT정책, 정보보호

**이 남 용** (Nam-yong Lee)

1980년~1983년 고려대학교 경영대학원 경영정보학(MIS)(경영학석사)
1990년~1993년 미국 미시시피주립대학교(MSU) 경영정보학(MIS)(경영학박사)
1999년 숭실대학교 컴퓨터학부 교수
<관심분야> 소프트웨어 테스팅, 품질보증, MIS, 정보보호

**김 종 배** (Jong-Bae Kim)

2002년 8월 숭실대학교 대학원 석사
2006년 8월 숭실대학교 대학원 박사
2006년 8월 숭실대학교 대학원 박사
2001년~2012년 (주)이엔터프라이즈 대표이사
2004년~2006년 남서울대학교 컴퓨터학과 겸임교수
2006년~2012년 서울여자대학교 컴퓨터학부 겸임교수
2012년~현재 숭실대학교 SW특성화대학원 교수
<관심분야> 소프트웨어 개발 방법론, 정보보호, 오픈소스소프트웨어