대용량 데이터 처리기술을 배전자동화 시스템에 적용

논 문
60-2-3

Application of Data Processing Technology on Large Clusters to Distribution Automation System

이 성 우† · 하 복 남* · 서 인 용* · 장 문 종*
(Sung-Woo Lee · Bok-Nam Ha · In-Yong Seo · Moon-Jong Jang)

Abstract - Quantities of data in the DMS (Distribution management system) or SCADA (Supervisory control and data acquisition) system is enormously large as illustrated by the usage of term flooding of data. This enormous quantity of data is transmitted to the status data or event data of the on-site apparatus in real-time. In addition, if GIS (Geographic information system) and AMR (Automatic meter reading), etc are integrated, the quantity of data to be processed in real-time increases unimaginably. Increase in the quantity of data due to addition of system or increase in the on-site facilities cannot be handled through the currently used Single Thread format of data processing technology. However, if Multi Thread technology that utilizes LF-POOL (Leader Follower -POOL) is applied in processing large quantity of data, large quantity of data can be processed in short period of time and the load on the server can be minimized. In this Study, the actual materialization and functions of LF POOL technology are examined.

Key Words : SCADA, RTU, FEP, Single thread, Multi thread

1. 서 론

자동화 시스템에서는 현장에서 전달 받은 데이터를 시스템을 사용하는 사용자에게 손실 없이 빠르게 전달하여 사용자가 꼭 알아야 하는 각종 정보들을 표현해 줄 수 있어야 한다. SCADA나 DMS 시스템에서는 데이터의 홍수라 해도 과언이 아닐 정도로 많은 양의 데이터가 실시간으로 현장 기기의 상태 데이터나 이벤트 데이터 등으로 전송된다. 또한 GIS, AMR 등이 통합 될 경우 처리해야 할 실시간 데이터는 더욱 늘어나게 된다.

기존 배전자동화 시스템에서는 Single Thread 방식의 데이터 처리기술을 사용하였다. 이 작업 방식의 이점은 하나의 Thread에서 모든 처리가 이루어지기 때문에 구현방법이 간단하고 시스템을 이해하기가 무척 쉽다. 하지만 하나의 작업을 처리하는 동안 다른 데이터의 처리가 불가능하기 때문에 동시에 여러 데이터가 발생할 경우 모든 데이터를 처리하는데 오랜 시간이 소모된다.

지속적인 현장 설비의 증가와 단말장치 기술 향상 및 시스템 추가에 따른 데이터양은 기존의 사용하고 있는 Single Thread 방식의 데이터 처리 기술을 통해서 감당하기 힘들 정도로 늘어나게 되었다. 따라서 대용량 데이터를 실시간으로 손실 없이 빠르게 처리 할 수 있는 기술 및 구현방법이 요구된다.

본 논문에서는 자동화 시스템에 있어서, 현장에서 발생하는 데이터들을 더 신속하게 처리하여 시스템을 사용하는 사용자가 정보를 빨리 제공받을 수 있도록 하는데 있다. 하나의 작업자를 두고 데이터를 처리하는 방식은 시간대비 작업 효율이 현저하게 떨어진다. Multi Thread 기술을 통한 동시 작업 처리는 데이터를 처리하는데 있어서 효율성을 극대화 할 수 있다.

Multi Thread 기술을 통해 데이터 큐에서 데이터를 처리하는 Thread를 만들어 작업을 동시적으로 처리하게 되면 성능 및 시간상으로 월등한 우위를 기대 할 수 있다. Multi Thread 기술을 적용했을 경우에 얻을 수 있는 이익으로는 첫째: 여러 작업자가 여러 작업을 독립적으로 동시에 처리할 수 있고 둘째: 서버급의 멀티 프로세스시스템에서는 운영체제가 각각의 CPU에게 Thread를 할당하며, 이에 따라 작업을 실시간으로 독립적으로 처리할 수 있다. 대용량 데이터 처리기술은 Thread 간의 동적 메모리 할당 최소화, 공유 데이터를 최소화한 동기화 객체 사용 최소화, Thread 간의 Context Switching을 유발 시킬 수 있는 요소 제거, 적절한 크기의 Thread 수를 유지하며 OS의 자원 사용 등에 대한 유의점들을 설계단계에서부터 적용하여 기술하였다.[1,2,3,4]

본론에서는 데이터 집결지 Queueing Layer에 대해서 설명하고, Single Thread 방식, Thread pre Each Data Processing 방식, Leader - Follower Thread Pool 방식에 대해 장단점을 비교분석하였다. 또한, LF-FOOL를 구현하기 위하여 LF-POOL 배치와 리더의 권한과 LF-POOL의 구성요소인 Follower Class와 LF-Pool Class에 대해 기술하였으며, LF기술을 빠른 속도를 필요로 하는 프로그램에 적용하기 위해 기능의 모듈화와 동적 다이아그램으로 표현하였다. 본 연구내용을 실계통 시스템인 배전자동화 스카다 통합시

† 교신저자, 정회원 : 한전 전력연구원 송배전연구소
E-mail : swlee@kepri.re.kr
* 정 회 원 : 한전 전력연구원 송배전연구소
접수일자 : 2010년 8월 23일
최종완료 : 2011년 1월 5일

스템에 적용하였으며, 실계통 시험은 1초당 처리 알람수와 총 처리시간을 비교한 시험결과를 제시하였다.

2. 본 론

2.1 데이터 집결지 Queueing Layer

여러 종류의 데이터는 다양한 루트에서부터 전송된다. DMS시스템에서는 현장의 개폐기 상세 정보 데이터들이 현장의 RTU(Remote Terminal Unit)에서 FEP(Front-end Processor)쪽으로 전송된다. 또한 변전소의 MTR 뱅크의 상태 정보 값은 MTR RTU에서 FEP로 전송 될 수도 있고, 별도의 에이전트를 통해 집적 Middleware로 전송된다. 이때 모든 데이터의 목적지는 데이터 큐로 전송된다. 빠른 데이터 처리의 핵심 사항은 데이터 큐에 쌓인 실시간 정보 데이터를 어떻게 처리 할 지부터 시작한다. 이 실시간 데이터는 History용 DB에 마지막 상태 정보를 기록 할 수도 있고, 구간 부하량의 통계를 내기위해 시간대 별로 RDB에 저장될 수도 있고, 사용자에서 실시간으로 알려야 하는 긴급한 이벤트 데이터가 될 수도 있다. 핵심사항은 어떻게 다양한 종류의 데이터를 실시간으로 처리 할 수 있는 기술을 개발하느냐 하는 것이다. 많은 양의 데이터를 실시간으로 처리하기 위해서 우리는 LF-POOL에서 멀티 쓰레드로 데이터를 처리하며 처리 할 데이터는 Queueing Layer에 저장되어 있다. LF-POOL은 주기적으로 폴링을 통해 데이터를 Queueing Layer에서 꺼내와 처리를 시작한다.

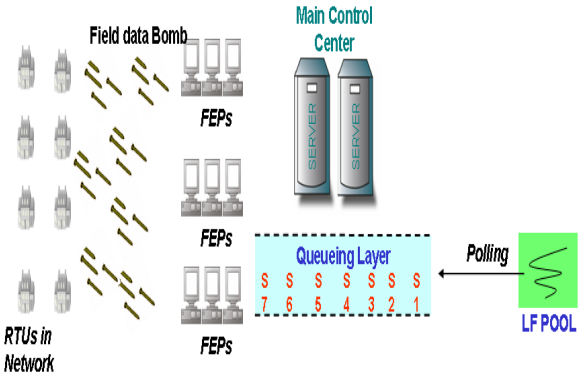

그림 1 실시간 데이터 집결지 Queueing Layer
Fig. 1 Queuing Layer, the collection site of real time data

2.2 싱글 쓰레드 방식과 멀티쓰레드 방식을 이용한 데이터 처리 기술

멀티 쓰레드 기술을 통한 동시 작업 처리는 데이터를 처리하는데 있어서 효율성을 극대화 할 수 있다. 하지만 멀티 쓰레드 기법을 통한 처리 기술은 다음과 같은 문제점을 가진다.

- 멀티쓰레드 기술을 통해 구현하는 방법은 구현이 상당이 어렵다.
- 시스템을 구현 시 OS에 관련해 동기화, 컨텍스트 스위칭 등의 기본기술을 이해해야 한다.
- 동기화를 잘못 구현 했을 경우 데드락 상태에 빠진다.
- 시스템에 대한 기본적인 데이터 흐름을 완전히 이해하고 설계, 구현해야 한다.

하지만, 멀티 쓰레딩 기술을 통해 데이터 큐에서 데이터를 처리하는 쓰레드를 만들어 작업을 동시적으로 처리하게 되면 성능 적으로나 시간적으로나 월등한 우위에 서게 된다. 다음은 멀티쓰레드 기술을 적용했을 경우 얻을 수 있는 이점이다.

- 많은 작업자가 여러 작업을 독립적으로 동시에 처리 할 수 있다.
- 서버급의 멀티 프로세스 시스템에서는 운영체제가 각각의 CPU에게 쓰레드를 할당하며, 이에 따라 작업을 실시간으로 독립적으로 처리 할 수 있다.

많은 양의 데이터를 처리 할 때 멀티 쓰레드 기술을 적용하는 방법은 다양하게 존재한다. 어떤 방법을 사용하느냐에 따라서 데이터를 처리 하는데 있어서 속도나 시스템에 어느 정도 부담을 줄 수 있는지 결과가 다양하게 존재한다.

데이터 처리 작업을 동시에 빠르게 처리 할 수 있는 능력을 가진 시스템을 구현하기는 상당히 어렵고 복잡하다. 빠른 성능을 보장하기 위해서는 다음과 같은 문제점들이 설계 단계에 미리 고려되어야 하며 최소화 시켜야 한다.

- 쓰레드 간의 동적 메모리 할당 최소화
- 쓰레드 간에 공유데이터를 최소화하여 동기화 객체 사용 최소화
- 쓰레드 간에 Context Switching을 유발 시킬 수 있는 요소 제거
- 적절한 크기의 쓰레드 수를 유지하여 OS의 자원을 사용

2.2.1 Single Thread 방식의 데이터 처리기술

싱글 쓰레드에서의 작업 처리는 Reactive 방식과 유사하며 작업 처리는 메시지 큐에서 데이터를 꺼내온 후 하나씩 데이터를 처리 하는 방식이다. 데이터를 처리한 후 다시 큐에서 데이터를 꺼낸 후 작업을 반복한다. 위의 방식에서의 이점은 하나의 싱글 쓰레드에서 모든 처리가 이루어지기 때문에 구현 방법이 간단하고 시스템을 이해하는데 있어서 아주 쉽다. 하지만 하나의 작업을 처리 하는데 DB쓰기 작업이라든가, 데이터 분석에 시간이 많이 걸리는 경우 큐에 있는 데이터를 모두 처리 하는데 있어서 시간적으로 상당히 오랜 시간이 소모된다.

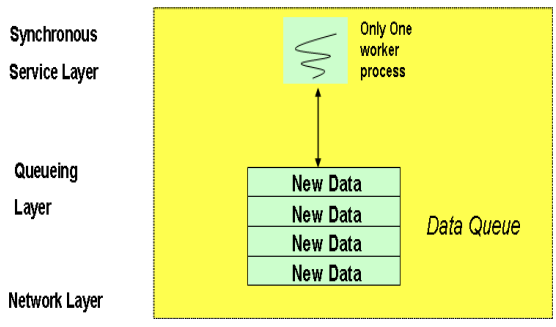

그림 2 싱글 쓰레드를 이용한 데이터 처리
Fig. 2 Data processing for Single Thread

위의 문제점으로 인해서 실시간 고성능 데이터 처리 시스템(High Performance Realtime Tranjection System)에서는 데이터 처리를 할 때 하나의 쓰레드를 두고 작업하지 않으며 성능상의 이점을 찾을 수 없다.

2.2.2 Thread pre Each Data Processing 방식의 데이터 처리 기술

Thread pre Each Data Processing 방식은 하나의 메인 쓰레드에서 데이터를 수집한다. 데이터를 수집하는 시점에 하나의 쓰레드를 생성하여 생성된 쓰레드에서 작업을 수행하는 방식이다. 이 방식은 구현 방법이 상당히 간단하며 쉽고 가장 널리 쓰이는 방식이다. 하지만 구현이 간단한 만큼 많은 양의 데이터를 효과적으로 처리하는데 있어서는 문제점을 가진다. 가장 큰 문제점은 큐에 대용량의 데이터가 적재되어 있다면 해당 데이터의 수만큼 데이터 처리 쓰레드가 필요하다. 이것은 곳 자원낭비이며, 동적 메모리 할당에 빈번한 메모리 할당은 시스템의 성능 저하로 이루어진다. 또한, 여러 작업이 동시로 진행되지만 싱글쓰레드로 데이터를 처리 하는 것과 비교 할 때 시스템의 성능적인 이익을 가져올 수 없다.

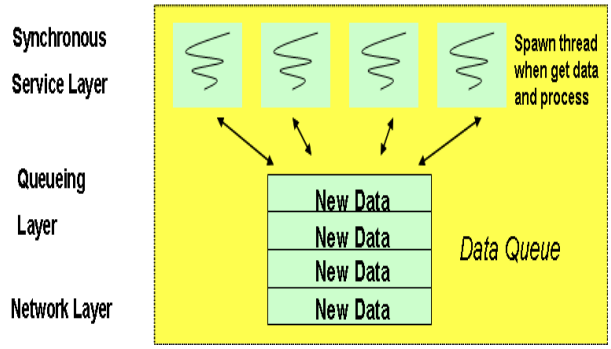

그림 3 한 트랜젝션 당 쓰레드를 할당한 데이터 처리
Fig. 3 Data processing to allocation of Thread per one Tranjection

2.2.3 Leader - Follower Thread Pool 방식을 이용한 데이터 처리

위의 Pool의 문제점은 Thread간에 과도한 Context Switching으로 인해 시스템의 성능을 저하시킬 수 있다는 점이다. 이러한 문제점을 해결하기 위해서는 데이터 큐에서 데이터를 획득 시 Context Switching을 제거해야 하며 그러기 위해서는 데이터를 획득하는 쓰레드의 수를 최소화하여 데이터를 처리해야 한다.

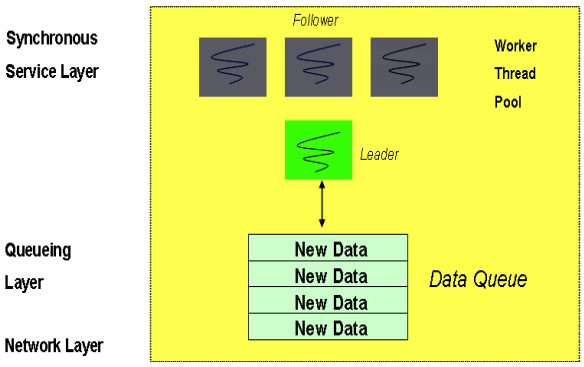

그림 4 LF POOL을 이용한 데이터 처리
Fig. 4 Data processing for LF POOL

이 문제를 해결하기 위한 기술은 데이터를 습득하는 하나의 쓰레드를 리더로 설정한다. 쓰레드 풀의 나머지 쓰레드들을 Follower라고 하며 자신이 리더가 되기를 기다린다. 이때는 오직 리더만이 데이터 큐에 접근하기 때문에 동기화 오버헤드나 Context Switching을 줄일 수 있다. 리더가 데이터를 꺼내는 동시에 데이터를 처리하며 리더의 자리를 Follower의 중 하나를 선택하여 리더를 채택한다. 리더의 권한을 가진 Follower의 데이터 큐에 접근하여 데이터를 처리하는 리더의 임무를 수행한다.

2.3 LF-POOL 구현

여러가지 기법을 분석한 후 대용량 데이터를 처리하는데 Leader - Follower Thread Pool을 이용하여 구현하였다.

2.3.1 LF-POOL 배치와 리더의 권한

기본적인 멀티쓰레드 기술을 이용하여 데이터를 처리할 경우 문제점은 Thread간에 과도한 Context Switching으로 인해 시스템의 성능을 저하시킬 수 있다는 점이다. 이러한 문제점을 해결하기 위해서는 데이터 큐에서 데이터를 획득 시 Context Switching을 제거해야 하며 그러기 위해서는 데이터를 획득하는 쓰레드의 수를 최소화하여 데이터를 처리해야 한다. 이 문제를 해결하기 위한 기술은 데이터를 습득하는 하나의 쓰레드를 리더로 설정하여 데이터를 처리하는 LF-POOL을 배치하여 데이터를 처리한다.

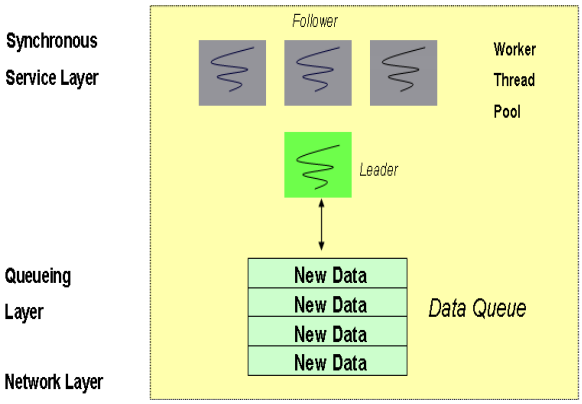

그림 5 LF-POOL 배치도
Fig. 5 Allocation Diagram for LF-POOL

LF-POOL 내부의 나머지 쓰레드들을 Follower라고 하며 자신이 리더가 되기를 기다린다. 이때는 오직 리더만이 데이터 큐에 접근하기 때문에 동기화 오버헤드나 Context Switching을 줄일 수 있다. 리더가 데이터를 꺼내는 동시에 데이터를 처리하며 리더의 자리를 Follower의 중 하나를 선택하여 리더를 채택한다. 리더의 권한을 가진 Follower는 데이터 큐에 접근하여 데이터를 처리하는 리더의 임무를 수행한다.

2.3.2 LF POOL의 구성요소

가. Follower Class

Follower는 LF POOL 내부의 대기자 쓰레드들이 리더가 될 때까지 쓰레드들의 제어를 멈추는 역할을 담당한다.

Follower Class의 역할로 인해 LF POOL에 존재하는 다수의 쓰레드들은 큐레이어의 폴링 처리를 리더가 될 때까지 기다릴 수 있으며 리더 쓰레드가 데이터를 큐에서 획득 후 데이터를 처리하기 전에 Follower Class에게 명령을 전달해 쓰레드의 대기를 멈춘다. Follow class의 선언과 구현한 내용을 간략히 설명하면 다음과 같다.

```
class Follower
{
public:
  Follower ()
 {
    owner_ = Thread::self ();
  }
  int wait (void)
  {
    return this->cond_.wait ();
  }
  int signal (void)
  {
    return this->cond_.signal ();
  }
Condition cond_;
};
```

Follow Class는 아래의 코드처럼 리더가 활동중이면 LF Pool내부의 코드에서 리더가 되기까지 대기한다.

```
LF_ThreadPool::become_leader (void)
  {
    뮤텍스 획득
    만약 리더가 활동중이면
      {
        Follower *fw = make_follower ();
        {
          while (leader_active ())
            fw->wait ();
        }
        delete fw;
}
```

나. LF Pool class

LF Pool은 동기화 객체를 공유하는 쓰레드들의 그룹이다. LF Pool내의 쓰레드들은 오직 하나의 쓰레드만이 큐 레이어에 접근하여 폴링하기 때문에 동기화로 인해 발생되는 컨텍스트 스위칭에 따른 과부하를 제거해준다. LF Pool 내부에서 쓰레드들 간의 위치 조정을 수행하여 어떤 쓰레드가 리더가 되는지 또한 리더가 데이터를 처리할 때 대기자 쓰레드 중 하나를 선출하여 리더의 자리에 위치시키는 역할을 담당하며 쓰레드의 수를 컨트롤 하여 시스템에 따라 적절한 쓰레드의 개수를 정한다. 다음은 LF Pool Class를 추상화한 클래스이다.

```
class LF_ThreadPool
{
public:
  LF_ThreadPool ();
private:
  int SVC();
  int become_leader (void);
  Follower *make_follower (void);
  int elect_new_leader (void);
  int leader_active (void)
  void leader_active (int leader)
  void process_message (MESSAGE *mb);
  int done (void)

private:
  int shutdown_;
  thread_t current_leader_;
  Mutex leader_lock_;
  Queue<Follower*> followers_;
  Mutex followers_lock_;
 };
```

위의 클래스에서 설명한 것과 같이 LF Pool 클래스의 핵심은 리더선출, 리더 자리에 쓰레드 위치, Follower 대기 유지, 데이터 처리라 말할 수 있다. 패킷 교환망에서, 전송할 데이터가 있을 때만 통신 경로 설정을 하고 통신을 끌낼 때는 선로 연결도 해지하는 통신 접속방식인 SVC(Switched Virtual Connection) 함수 내부에서 이 모든 과정이 이루어지며 SVC 내부 구현은 다음과 같다.

```
int LF_ThreadPool::svc (void)
{
  printf("LF_ThreadPool::svc\n");
  while (!done ())
    {
      become_leader ();  // Block until this thread is the leader.
      큐에서 데이터를 획득한다.
      데이터를 획득했다면,
      elect_new_leader(); 리더를 선출한 후
      process_message (Message);데이터를 처리한다.
    }
  return 0;
}
```

2.3.3 동적 다이어그램

SVC 내부에서 리더 위치 결정과 다른 쓰레드들의 대기, 리더가 데이터를 얻은 후에 새로운 리더의 선출, 데이터 처리가 모두 이루어진다. 다음은 위 작업이 수행되는 흐름을 활동 다이어그램으로 표현하였다.

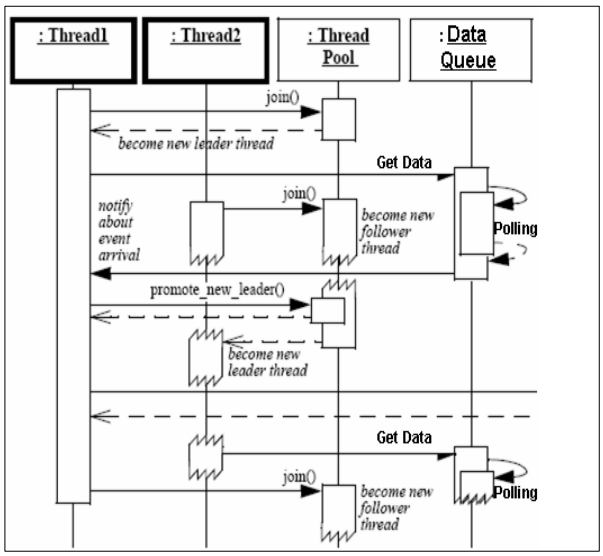

그림 6 동적 다이어그램
Fig. 6 Dynamic Diagram

2.3.4 LF-POOL 모듈화

LF기술은 빠른 속도를 필요로 하는 프로그램에 적용하기 위해 기능의 모듈화 작업이 필요하다. LF_MODULE는 WIN32 환경의 MSVC를 이용해 개발되었고 윈도우 환경에서 동작한다.

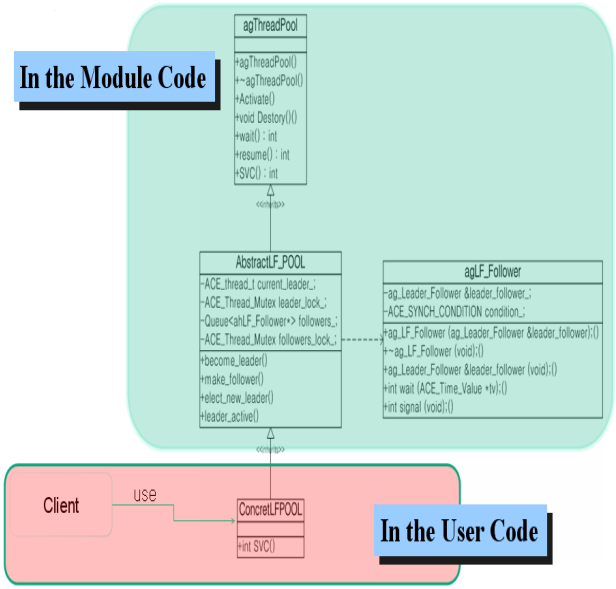

그림 7 LF-Pool 모듈화
Fig. 7 Modulization of LF-Pool

그림 7과 같이 모듈을 사용하기 위해서는 추상 LF_POOL을 상속받아 필요한 부분을 구현하면 모듈에서 LF의 기본 작업을 수행해 준다. LF_POOL기술은 다양한 분야에 적용될 수 있다.

그림 8은 FEP에 LF 모듈을 탑재한 기술을 적용한 예이다.

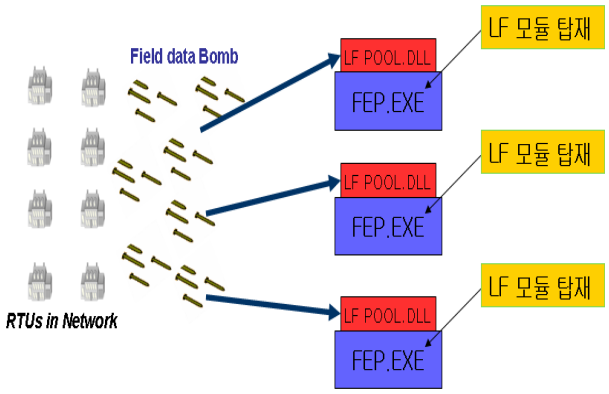

그림 8 FEP에 LF 모듈 탑재
Fig. 8 Loading of LF module onto FEP

FEP 프로그램은 LF_POOL.DLL을 탑재하여 현장에서 오는 데이터를 빠르게 처리할 수 있다.

2.4 배전 자동화 SCADA 통합 시스템에 적용

ScadaHost 프로그램에 LF POOL 기법을 적용하여 메시지 큐에 쌓여 있는 변경된 데이터를 다수의 쓰레드에서 동시 처리하여 많은 데이터가 수집되더라도 빠르게 대량의 데이터를 처리 할 수 있도록 적용하였다. 기술을 적용하기 위한 구성요소 컴포넌트는 다음과 같다.

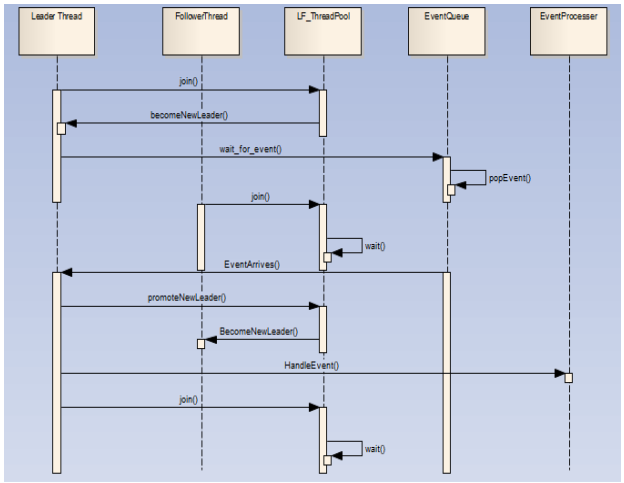

그림 9 LF POOL을 이용한 데이터 처리 과정 (UML 다이어그램)
Fig. 9 Data processing stage for LF POOL(UML Diagram)

2.4.1 구성요소

- Leader Thread - 이벤트 큐에서 데이터를 꺼내 처리
- Follower Thread - Leader가 될 때 까지 대기
- LF Thread Pool - 다수의 쓰레드를 관리하며 Leader 쓰레드와 Follower 쓰레드를 관리
- EventQueue - RTU에서 수집된 데이터가 적재, Leader는 이곳에서 데이터를 꺼냄
- Event Processor - 이벤트를 꺼낸 후 실제 적인 작업 처리를 한다. 데이터 비교, DB 쓰기, 알람 발생

2.4.2 시험 환경 구성

스카다 호스트 프로그램에서는 LF 기술을 모듈화 하여 RTU에서 전송된 데이터를 처리하는데 사용된다. 성능을 테스트하기 위해 가상으로 데이터를 만들어 이벤트 큐에 데이터를 넣은 후 방식별로 얼마나 시간이 소요되는지 측정하였다. 시험환경은 SCADA 이중화 서버, HMI, FEP, RTU 시뮬레이터 2대로 구성하여 시험하였다.

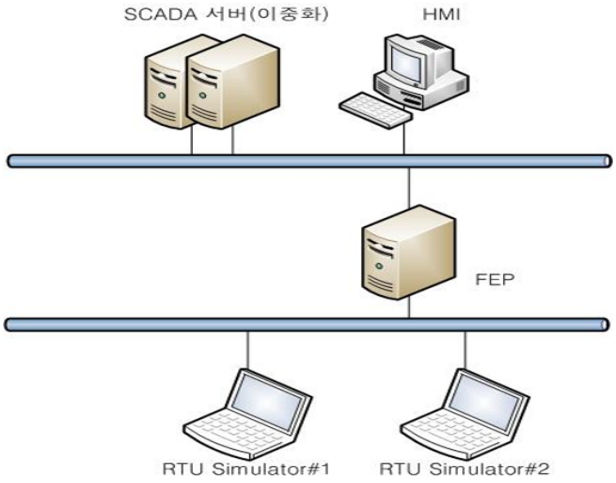

그림 10 시험 환경 구성
Fig. 10 Test environment constitution

2.4.3 시험 결과

시험 방식은 각각의 RTU Simulator에서 1000개의 데이터를 보내며 총 10개의 가상 RTU에서 데이터를 전송하였다. 데이터 처리 방법은 Single Thread 방식, Thread pre Each Data Processing 방식, Leader - Follower Thread Pool 방식을 이용해 1초당 처리 알람수와 총 처리시간을 비교하여 시험결과를 제시하였다.

표 1 시험 결과
Table 1 Test Result

데이터 처리 방식	1초당 처리 알람 수	총 처리시간
Single Thread 방식	400개	25초
Thread pre Each Data Processing 방식	378개	30초
Leader - Follower Thread Pool 방식	900개	11초

시험 결과는 표 1에서 보는 것과 같이 데이터 처리 방식 중에 Leader - Follower Thread Pool 방식이 1초당 처리 알람수가 900개, 처리 시간은 11초로 가장 우수한 성능을 보여주었으며, HMI에서 수신된 알람 데이터를 조회할 수 있다.

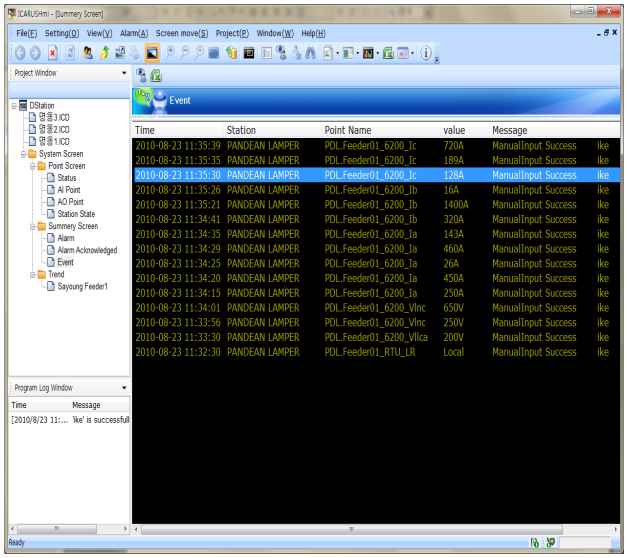

그림 11 HMI 에서의 알람 수신 화면
Fig. 11 Alarm received view of HMI

3. 결 론

대용량의 데이터를 처리하는데 있어 여러 가지 기술이 있지만, 가장 효율적인 방법은 LF Pool을 이용하여 데이터를 동시적으로 처리하는 것이다. LF기술은 CPU의 성능을 극대화 시키며, Thread간의 동적 메모리 할당에 따른 부하와 동기화에 따른 오버헤드를 효과적으로 감소시켜 실시간으로 처리하는데 우수한 성능을 보여준다. 또한 Thread의 데이터 습득에 있어 오직 리더만이 데이터를 얻을 수 있으므로 동기화에 대한 부하를 감소시킨다. 이에 따라 Thread간에 Context Switching도 일으키지 않으므로 고성능의 데이터 처리를 할 수 있다. 스카다 호스트 프로그램에 본 기술을 적용하여 현장에서 발생한 알람에 대해서 사용자의 HMI 화면에 지연 없이 보여 표현하기 때문에 스카다 시스템과 같은 실시간 시스템에 유용하게 사용되었다.

감사의 글

본 연구는 2008년도 지식경제부의 지원에 의하여 이루어진 연구로서, 관계부처에 감사드립니다.

참 고 문 헌

[1] 이성우외, “배전지능화 시스템 중앙제어장치 개발” 1단계 최종보고서”, 지식경제부, 2008
[2] 이미영, “클라우드 기반 대규모 데이터 처리 및 관리 기술”, 전자통신동향분석, 한국전자통신연구원, 41-54, 2009.
[3] 문현정외, “대용량 RDF 데이터의 처리 성능 개선을 위한 효율적인 저장 구조 설계 및 구현”, 한국전자거래학회지 12(3) 251-268, 2007.
[4] 이성우, “배전지능화시스템 중앙제어장치 개발”, 기술세미나, 한국전력공사 전력연구원, 2007.

저 자 소 개

이 성 우 (李 聖 雨)

1960년 3월 1일생(음). 1999년 건국대 대학원 전기공학과(공박), 1992년 전력연구원 입사. 1992~2006년 한전 전력연구원 발전연구소 근무. 2007년~현재 한전 전력연구원 송배전연구소 배전IT 부장. 배전지능화 및 배전 IT 시스템 분야 연구.
Tel : 042-865-5931
E-mail : swlee@kepri.re.kr

하 복 남 (河 福 男)

1958년 1월 10일생. 1994년 충남대 대학원 전기공학과(석사). 2004년 충남대 대학원 전기공학과(공박), 1978년 한국전력공사 입사 이후 대전전력관리처, 광주전력관리처, 전력연구원 근무, 1988년~현재 한전 전력연구원 송배전연구소 배전IT 처장.
Tel : 042-865-5930
E-mail : bnha@kepri.re.kr

서 인 용 (徐 寅 勇)

1961년 1월 13일생. 1989년 부산대학교 전기공학과(석사), 2000년 브라운대학교 응용수학과(석사), 2003년 브라운대학교 전기공학과(공박), 1984년 한국전력공사 입사 이후 고리원자력본부, 전력연구원 근무, 2000년~현재 한전 전력연구원 송배전연구소 배전IT 책임연구원. 모델링&시뮬레이션 분야 연구
Tel : 042-865-5932
E-mail : iyseo@kepri.re.kr

장 문 종 (蔣 汶 宗)

1970년 4월 24일생. 1995년 경북대 컴퓨터공학과 졸업. 1997년 KAIST 컴퓨터공학과 졸업(석사). 현재 한국전력공사 전력연구원 선임연구원
Tel : 042-865-5984
E-mail : mjjang@kepco.co.kr