Data Processing Technology on Large Clusters of Automation System

Sung-Woo Lee*, Bok-Nam Ha, In-Yong Seo, Moon-Jong Jang, Chang-Hoon Shin
Distribution IT
KOREA ELECTRIC POWER RESEARCH INSTITUTE
Daejeon
Korea
swlee@kepco.co.kr

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: - Automation System, DAS, Data Processing

1 Introduction

In automation system, one must be able to express various information that users must be aware of by expeditiously transmitting the data received from the site to the users that utilizes the system without losses in the data. In the SCADA or DMS system, flood of data is being transmitted to the status data or event data of on-site apparatus in real time. In addition, if GIS and AMR are integrated, the quantity of real time data to be processed increases further.

In the existing power distribution automatic system, Single Thread format of data processing technology is being used. Advantage of this processing method is that the materialization method is simple and it is very easy to understand the system since all processing is accomplished at a single Thread. However, while one task is being processed, it is impossible to process other data. Therefore, it takes long time to process all the data if several data are generated simultaneously.

The quantity of data due to continuous increase in on-site facilities, improvement of terminal apparatus technology and addition of system has increased to the level that is difficult to handle through the currently used Single Thread format of data processing technology. Therefore, technology and materialization method that can expeditiously process the large quantity of data in real time without losses of data is required.

This Study aims to enable the user of the automation system to receive information quickly by processing the data generated at the site more expeditiously. The format of processing the data with 1 operator will result in substantially low work efficiency in terms of time taken. Simultaneous processing of tasks through Multi Thread technology can maximize the efficiency in processing data.

If the tasks are processed simultaneously by creating Thread that processes the data at the data queue through Multi Thread technology, outstanding superiority in terms of performances and time can be expected. Benefits that can be attained through application of Multi Thread technology include the following: Firstly, several operators can independently processes several tasks simultaneously; Secondly, operating system at the multi-processor system at server level allocates Thread to each CPU, and tasks can be processed independently and in real time. Technology for processing of large quantity data minimizes the allocation of dynamic memory between Threads, minimizes the usage of synchronization object for which shared data has been minimized, removal of element that can induce Context Switching between Threads, and maintains the appropriate size of Thread number. This Study describes application of issues that requires attention regarding usage of resources of OS from the designing stage [1, 2, 3 & 4].
In the main discourse, data collection site, the Queuing Layer, is explained, and describes the technology of processing data by using multi-thread technology as well as ensuing problems. In addition, it explains the Follower Class and LF-Pool Class, which are the compositional elements of Frame Format and LT-POOL, used in embodiment of LF-FOOL, and modulization and dynamic diagram of functions for application of LF technology onto programs that require high speed were expressed.

2 Main Discourse

2.1 Queuing Layer, the collection site of data
Various kinds of data are transmitted from wide range of routes. In the DAS system, detailed information data on the on-site switch are transmitted from the on-site RTU to FEP. In addition, status information values of MTR bank of substation can be transmitted from MTR RTU to FEP, or accumulation Middleware through separate agent. At this point, all data are transmitted to the data queue. The core issue in expedient process of data begins with how the real time information data accumulated at the data is processed. This last status information on this real time data can be recorded in the DB used for History, can be stored in separate RDB for each time interval in order to establish statistics on the load for each sector, or can become urgent event data that must be notified to the user in real time.

As such, the core issue is how to develop technology that can process diverse range of data in real time. In order to process large quantity of data in real time, data was processed with multi-thread at the LF-POOL, and the data to be processed was stored in the Queuing Layer. LF-POOL periodically takes the data out of the Queuing Layer through pooling and begins processing of the data.

2.2 Data processing by using multi-thread technology
Simultaneous task processing through multi-thread technology can maximize the efficiency in processing data. However, processing technology through multi-thread technique has the following problems.

- Method of embodiment through multi-thread technology is extremely difficult to realize.
- One must understand basic technology including synchronization and context switching in relations to OS at the time of embodiment of the system.
- One will end up with dead-lock if realization of synchronization fails.
- Designing and realization must be carried out with complete understanding of the fundamental flow of data on the system.

However, if tasks are processed simultaneously by creating thread that processes the data at the data queue through multi-threading technology, one will be able to obtain functional and temporal superiority. Following are the advantages one can gain when multi-thread technology is applied.

- Many operators can independently process various tasks simultaneously
- Operating system in the multi-process system at server level allocates thread to each CPU, and accordingly, tasks can be processed independently in real time

There is diverse range of methods of multi-thread technology application when processing large quantity of data. Depending on the method applied, wide range of results of the extent of load imparted onto the speed or system in processing data exist.

It is very difficult and complicated to realize system with ability to perform data processing tasks simultaneously and expediently. In order to ensure expedient function, the following problems must be considered in advance and minimized at the design stage.

- Minimization of allocation of dynamic memory between threads
- Minimization of usage of synchronization object by minimizing the shared data between the threads
- Removal of elements that can induce Context Switching between threads
- Usage of resources of OS by maintaining appropriate size of thread number
In considerations of above problems, LF-POOL optimized for processing of large quantity data was realized.

2.3 Realization of LF-POOL

2.3.1 Frame Format

Problem in processing of data by using the basic multi-thread technology is that it can lower the functions of the system due to excessive Context Switching between the threads. In order to solve such problem, Context Switching must be removed at the time of acquisition of data from the data queue. For this purpose, data must be processed by minimizing the number of threads that acquire data. Technology used to solve this problem processes the data by allocating LT-POOL that processes the data by setting one thread that acquires data as the leader.

![Figure 2 ALLOCATION DIAGRAM FOR LF-POOL](image)

The remaining threads within the LF-POOL are referred to as the Followers and they wait to become the leader. Since the only the leader accesses the data queue at this time, it is possible to reduce the synchronization overhead or Context Switching. The leader processes the data as soon as it takes the data from the data queue, and selects one of the followers to assign the status as a leader to the selected follower. The follower to which the authority as a leader has been assigned carries out the role of leader of processing the data by accessing the data queue.

2.4 Compositional elements of LF POOL

2.4.1 Follower Class

Follower plays the role of stopping the control of threads until the threads in stand-by within the LF POOL becomes the leader. Several threads existing in the LT POOL due to the role of Follower Class can wait for polling processing of the queue layer until it becomes the leader, and stops the standing-by of the thread by conveying the command to the Follower Class before the leader thread processes the data after having acquired data from the queue. Brief explanation of the declaration and details of embodiment of Follow class is as follows:

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

Follow Class, if the leader is active as illustrated in the following code, waits until it becomes the leader within the code within the LF Pool.

```cpp
LF_ThreadPool::become_leader (void)
{
  Acquisition of Mutex
  If the leader is active
  {
    Follower *fw = make_follower ();
    while (leader_active ())
      fw->wait ();
    delete fw;
  }
```

2.4.2 LF Pool class

LF Pool is the group of threads that shares the synchronization object. Because only one of the threads within the LF Pool carries out polling by accessing the queue layer, it removes the excessive load arising from context switching generated due to synchronization. When deciding which thread become the leader and when the leader processes the data while coordinating the location between the threads within the LF Pool, decide on the appropriate number of threads in accordance with...
the system by controlling the number of threads through taking up of the role of electing one of the threads on stand-by and assigning the status as the leader. The following is a class that abstracts the LF Pool Class.

class LF_ThreadPoll
{
    public:
    LF_ThreadPoll();
    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_;    

    As explained above, the core of the LF Pool Class includes election of the leader, assigning of the status of leader to thread, maintenance of standing-by of Follower and data processing. All of these processes are accomplished within the SVC function, and internal realization of SVC is as follows.

    int LF_ThreadPoll::svc(void)
    {
        printf("LF_ThreadPoll::svc\n");
        while (!done())
        {
            become_leader(); // Block until this thread is the leader.
            Acquire data from the queue
            Once the data is acquired
            elect_new_leader(); after having elected leader
            process_message (Message); process data.
        }
        return 0;
    }

2.5 Dynamic diagram

Decision on the location of the leader, standing-by of other threads, election of new leader after the leader has acquired data, and processing of data are all accomplished within the SVC. The following diagram expresses the flow of the execution of aforementioned tasks.

2.6 LF-POOL modulization

Modulization of the function is necessary in order to apply LF technology to program that requires fast speed. LF_MODULE was developed by using MSVC under WIN32 environment, and is operated under Window environment.

In order to use the module as per the Figure 4, realize the necessary aspects by inheriting the abstracted LF_POOL in order to carry out the basic tasks of LF at the module. LF_POOL technology can be applied to diverse range of fields.

Figure 5 illustrates the example of technology in which LF module has been loaded onto the FEP.

FEP program can expediently process the data arriving from the site by lading LF_POOL.DLL.
3 Conclusion

Although there are various technologies available for processing of large quantity of data, the most efficient method is to process the data simultaneously by using the LF Pool. LF technology maximizes the performances of CPU, and displays outstanding performances in processing the data in real time by effectively reducing the load generated by allocation of dynamic memory between the threads and overhead arising from synchronization. In addition, since only the leader can acquire data in acquisition of data by Thread, it reduces the load on synchronization. Accordingly, data can be processed with high level of performance as there will be no Context Switching between the threads either.

4 ACKNOWLEDGMENT

This work was supported by the Korea Institute of Energy Technology Evaluation and Planning (KETEP) grant funded by the Korea government Ministry of Knowledge Economy. (No. 2008T100100236).

References: