Priority-based Parallel Processing Scheme of Mass Data for Real-time DDS Monitoring System

Min-Young Son
ICT Convergence Research Center
Kumoh National Institute of Tech.
Gumi, Korea
Son0804@kumoh.ac.kr

Joong-Hyuck Cha
ICT Convergence Research Center
Kumoh National Institute of Tech.
Gumi, Korea
JH.Cha@kumoh.ac.kr

Dong-Seong Kim
Dept. of IT Convergence Engr.
Kumoh National Institute of Tech.
Gumi, Korea
dskim@kumoh.ac.kr

***Abstract*—** This paper proposes a priority-based monitoring system in Data Distribution Service environment where messages are frequently transmitted and received. The proposed system uses parallel processing and priority-based algorithm to perform monitoring and message error detection in real-time. The proposed system is described by algorithm and its performance is verified through simulation.

Keywords—DDS; Large DBMS; Parallel Processing; Priority Scheduling

I. INTRODUCTION

Data Distribution Service (DDS) is an Object Management Group (OMG) standard that simplifies complex network programs and is useful for reliable, real-time data communication. DDS data communications can be executed regardless of the location or existence of applications [1,2]. In the DDS, messages are designed to be disposable, but in some cases, it may be necessary to store them for various purposes [3]. One of these goals is that many developers share the same components. As a developer updates the status of a message with a component as it progresses, developers use an application that can store and manage messages using the same development interface.

Another purpose is to monitor the communication state of the DDS. The messages are collected for monitoring, but a large number of messages cannot be processed in real-time on large systems. This problem can be solved by using a statistical algorithm after the system has saved the message. In addition to simply monitoring, data storage can be used to detect and respond to errors such as message contention and network errors [4].

It is difficult to deal with error detection or message corruption in current DDS systems because of DDS structural nature [5]. However, even small errors in messages can cause fatal problems in certain cases, such as combat systems, avionics systems. It is important to detect and respond errors in messages. However, there is not much research on the processing of DDS monitoring system and error detection. Thus, this paper proposes an efficient system to monitor the transmission and reception of DDS messages and detect and cope with errors.

II. PROPOSED PRIORITY-BASED PARALLEL PROCESS

A. Parallel Process

In a DDS structure such as a combat system for collecting data from a plurality of sensors, a large amount of data is generated in a short time. Bulk-generated messages can cause data collisions, reduce real-time performance, and cause loss [4,5,7]. Therefore, this paper proposes a parallel multistep processing method to reduce this possibility. The data received from the data reader is distributed to the parallel processors by the scheduler, as shown in Figure. 1, and each processor performs the assigned tasks. Messages processed by each DDS Listeners are sent to the monitoring scheduler from DCPS layer. Messages include DDS entities which are QoS, message sample, Topic, timestamp, etc. in DDS Listeners, defined DCPS 1.2v by OMG [5] in DCPS. The message scheduler is only responsible for assigning messages to parallel processors.

The number of parallel processors in the DDS monitoring system should be designed considering the average number of messages per second and message processing time. If the average number of messages is 100 and the time required for processing is an average of 1 second, you must have a minimum of 100 processors to set up a real-time standby real-time environment. In actual implementation, parallel

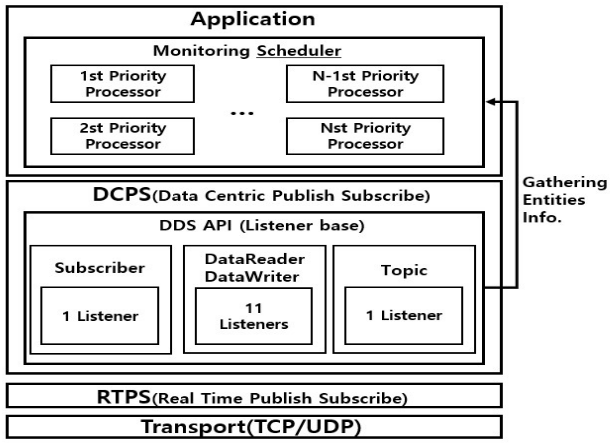

Figure 1. Proposed parallel process for real-time DDS monitoring system

environment can be implemented by GPU or OpenCL, and it can be implemented in cloud pc environment. This needs to be determined by considering the purpose of the monitoring program and the pros and cons of each parallel environment.

B. Priority of Processor

Among the errors in the messages that can appear in the DDS, defects in the structure of themselves can be relatively easily detected. However, in a normal message structure, it is difficult to immediately detect an error or malicious message change in a message. In some cases, data mining based on existing data can be useful to identify messages with a high probability of error. Data mining requires high time in case of complex algorithms, the immediate implementation of additional operations in DDS, which must guarantee real-time performance, causes various side effects. Thus, it is more efficient and safer to perform data mining with low priority after storing data and performing basic operations first [6].

Data Structure I.

```
Typedef struct Message{
  Data data;
  Int level;
}Message;
Message Queue[3];
Typedef struct Token{
  Boolean status;
  Int level;
  DateTime start;
}Token;
Token t[Processor_Count];
```

In proposed system, high priority messages are preferentially assigned to processors. The parallel processor that has been assigned the message performs the appropriate step operation according to the priority. At this time, the priority of the message is included in the message, and the data structure for the message is Structure I. The lower the value at the message level, the higher the priority. In this study, main-priority tasks can be divided into three stages, and the scheduler processes three queues corresponding to each priority.

Algorithm I is what the scheduler will do when it receives a message from the DataReader listener. First, the scheduler changes the message structure to include the priority of the message. The scheduler checks the tokens of all processors to see if there are empty processors. If all processors are running, the message is inserted into the highest priority queue and waits for the processor to return the token. If the queue is full, the process with the lowest priority and just started is forcibly stopped. At this time, the stopped job is saved.

Algorithm I. MessageReceiveFromDataReader

```
Input : data
 Create Message={data,0};
 Boolean flag=false;
 For i = 0 to Processor_Count-1 do
   If t[i].status == true then
      Processor[i].TokenSend(Message);
      flag =true;
      Break;
  If flag == false then
   If Queue[0] is full then
     i=find_most_high_level_and_closest_start_processor_idx
     save(Processor[i]. quit());
     Processor[i].TokenSend(Message);
   else Inqueue(0,Message);
```

The communication between the scheduler and the parallel processor is performed as in Algorithms 2 and 3. Algorithm II is called when the processor completes an operation on the assigned stage and returns a token to the scheduler. The scheduler sets the message level to the next level, examines the queue, assigns the task, and assigns the next task to be performed by the process that returned the token in order of priority. Algorithm 3 is called when the processor is assigned a task from the scheduler. Different tasks are performed depending on the level of the given message and the tokens are returned to the scheduler when the task is finished.

Algorithm II. TokenReceiveFromParallelProcessor

```
Input : processor's no and Message
 t[no].status = true;
 If message.level<2 then
   message.level++;
   inqueue(Queue[message.level], Message);
 For i<-0 to 2 do
   If Queue[i] is empty then  Load(i);
   If Queue[i] is not empty then
     Processor[no].TokenSend(dequeue(Queue[i]));
     t[no].status = false;
     t[no].level = message.level;
     t[no].start = now();
     Break;
```

The first of the three priority tasks is a mandatory or default action for the monitoring system. The basic task is to receive and store data, update required parameters, and perform manual monitoring. In the second priority, a relatively simple error is detected. At this priority, it is possible to check whether the data value is different from the average, abnormal transmission/reception time, message length error, and so on. At the final priority, the message is analyzed using a variety of data mining techniques. This takes a long time, but it can detect errors that cannot be detected in the previous step.

III. Simulation and Numrical results

A. Simulation Environment

The proposed system is simulated using C++. The simulation was performed on an Intel Core i5-7400 3.0GHz PC with 8GB of RAM. The simulation tests the performance of the proposed method by measuring the latency of each message in order to process the message in this situation. In this simulation, the monitoring system receives an average of 500 messages per second. It is assumed that the monitoring system suddenly receives 1,000 to 5,000 messages in a second while receiving an average message. In the simulation, the number of parallel processors is assumed to be 1,000.

In the simulation, the execution time of the three tasks corresponding to each priority is generated by generating a random number that follows the normal distribution of the mean and standard deviation, as shown in Table I. The execution time in the actual monitoring system will be determined differently depending on the purpose of the monitoring program.

Three cases to compare performance with the proposed system are "Only basic monitoring tasks", "Basic process + Simple fault checking" and "All monitoring tasks".

TABLE I. SIMULATION PARAMETER

Process	Execution Time	
	Mean (ms)	*Std (ms)*
Basic Process	100	10
Simple Fault Check	200	50
Data Analysis	400	300

B. Simulation Results

The proposed system and the comparator were executed 10 times according to the case, and the average value of processing time the message was measured as shown in Table 2. Experimental result show that high processing time often occurs in comparison systems. This is because fault checking, and data mining techniques take relatively long execution time.

TABLE II. AVERAGE OF PROCESSING TIME ALONG PROCESS KIND

	Message Number	Average of processing time (ms)			
		The Proposed System	*Basic Process*	*Basic+ Simple Fault Checking*	*All Monitoring Tasks*
Basic Process	1000	1	1	1	1
	2000	15	14	48	221
	3000	28	21	87	398
	4000	37	29	153	666
	5000	53	45	199	850
Simple Fault Checking Process	1000	115	-	98	97
	2000	135	-	143	399
	3000	188	-	186	812
	4000	288	-	268	1398
	5000	330	-	301	1733
Data Analysis Process	1000	260	-	-	256
	2000	776	-	-	763
	3000	1123	-	-	1098
	4000	1865	-	-	1680
	5000	2566	-	-	2485

The basic process lines in Table II show the average of processing time for the basic process in the monitoring system. Because this simulation deliberately generates excessive messages, the monitoring system must be able to handle messages well in high- traffic situations. As a result, the proposed system did not show a performance degradation compared to a comparative environment that performs basic data analysis, even though the proposed system also performs in-depth data analysis functions. The processing time is shorter that of the comparison environment, which includes the basic process and simple fault checking.

The simple fault checking process lines in Table II explain the average of processing time for the simple fault-checking process. The environment that contains the basic process is excluded from comparison because it does not perform simple fault checking. Considering that the difference in latency is not large and the proposed system provides all functions, the proposed system has superior performance compared to the function.

The data analysis process lines in Table II show the average latency for the data-analysis process. The simulation results show that the processing time of the proposed system is slightly longer than that of the comparison environment. This is because the proposed system executes the higher-priority processes first. This improves the real-time performance of the monitoring system.

IV. CONCLUSION AND FUTURE WORKS

The proposed system is for efficient monitoring and error detection in a DDS environment in which a large number of messages are transmitted and received. For monitoring and error detection, message processing is handled in parallel in order to maintain the real-time characteristics of the DDS. To avoid compromising the real-time characteristics of DDS, the monitoring functions are grouped into three priorities according to their importance, and separate queues are operated according to priority to allocate tasks to parallel processors. Simulation results show that the system creates an environment for in-depth analysis without compromising real-time characteristics and displays an immediate response to a large number of messages. It is necessary to implement the proposed system in the future and test the error detection scheme through system verification. As a future work, the data-mining techniques and algorithms will be combined with the proposed system.

ACKNOWLEDGMENT

This work was supported by Priority Research Centers Program(2018R1A6A1A03024003) and Advanced Research Project (NO. NRF-2017R1A2B4009900) through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology.

REFERENCES

[1] Min-Young Son, Dong-Seong Kim, and Joong-Hyuck Cha, "Efficient DDS monitoring system for large amount of data," 2018 14th IEEE International Workshop on Factory Communication Systems (WFCS), June 2018.

[2] Dong-Seong Kim and Sung-Kil Huh, "Distributed Control Networks of Naval Combat Systems," Journal of KIICE, vol. 13, no. 2, pp. 41-47, 2012.

[3] Gunjae Yoon, Jungwoo Choi, Huihwan Park, and Hoon Choi, "Topic naming service for DDS," 2016 International Conference on Information Networking (ICOIN), Jan. 2016.

[4] Hyun-Ji Kim, Ok-Kyoon Ha, Yong-Kee Jun, and Hee-Dong Park, "Message Races in Data Distribution Service Programs," 2015 8th International Conference on Database Theory and Application (DTA), Nov. 2015.

[5] Joong Hyuck Cha and Dong Seong Kim, "Design and Implementation of Real-Time Monitoring Tool for Data Distribution Service," IEIE Transactions on Smart Processing and Computing, Vol. 7, No. 4, pp. 264-270, Aug. 2018

[6] Williams Paul Nwadiugwu, Joong-Hyuck Cha, and Dong-Seong Kim, "Enhanced SDP-dynamic bloom filters for a DDS node discovery in real-time distributed systems," 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Sept. 2017.

[7] Muhammad Rizal Khaefi, Jin-Yong Im, and Dong-Seong Kim, "An efficient DDS node discovery scheme for naval combat system," 2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA), Sept. 2015.