A Real-time data distribution scheme of MOMAT for the Naval Combat System

Maksumov Bakir, Dwi Agung Nugroho, Yang Wei and Dong-Seong Kim
Networked Systems Lab, School of Electronic Engineering
Kumoh National Institute of Technology, South Korea
Email: {bakir, dwi.agung, yangwei, dskim}@kumoh.ac.kr

***Abstract*—The paper discusses MOMAT (Message Orientated Modeling and Analysis Tools) which is designed and implemented as middleware for performance of message oriented modeling and analysis tools for naval combat system (NCS). NCS consists of various large-scale components such as communication service and data distributed service (DDS). Furthermore, MOMAT meets requirements of NCS to support several versions of control such as components, interfaces, messages and publisher/subscriber paradigms. The MOMAT dealing with real-time data changes using continuous query (CQ) to provide new result from the database without issue same query repeatedly. However, a large number of continuous queries may be executed simultaneously over a high throughput of incoming data that may deteriorate system performance and cause data (message) delay. Therefore paper proposes a query split mechanism for performance improvement of retrieval in middleware systems.**

The implementation result shows that our proposed query-split management scheme significantly reduce workload of database server and increase scalability of each application.

***Index Terms*—Data Distributed Service (DDS), MOMAT, Query-split management scheme, MySQL database system, continuous query, NCS.**

I. Introduction

The main purpose of warships is to accomplish the assigned task by eliminating threats and attacks during the mission time. For defense and control warships, it uses modern naval combat system (NCS) such as: sensors, weapons, fire control systems, guided missile and torpedo launchers as well as helicopter support system. NCS provides an information for all different situations in case of unpredictable enemy attack [1].

Middleware is one of important instrument to manage a lot of heterogeneous and multiple data in real-time. Especially, data distribution service (DDS) which is usually used in message oriented environments. In DDS, each node has priority to publish-subscribe between DDS fundamental components. To develop large-scale systems, many developers participate in various projects and cooperate each other. Large-scale systems are constituted of numerous components which depend on configuration of NCS [2][3][4][5]. For example, several developers have implemented numerous components to NCS. During the procedure of implementation, if changes are not properly applied to component, it is suboptimal to support the version of each component. Therefore, it can cause several problems that decrease the development efficiency. To cover these problem, message orientated management and analysis tool (MOMAT) is designed and implemented to satisfy following requirement.

However, this application has some couple of problems such as low scalability and limited ability of application. To process a transaction continuously, the system needs to handle big number of information due to the scale of the network. To address this issue, the paper propose query-split management scheme that adopted split and group operator to split queries into small groups for small groups to improve the performance of data retrieval in pub/sub system.

The remainder of this paper is organized as follows. Section II briefly explains message orientated modeling and analysis tool's (MOMAT) architecture and its performance. Section III presents the theory of query-split management scheme and its algorithm. Section IV present implementation and performance analysis. Finally, we present conclusion and future work in Section V.

II. System Model

A. Design of MOMAT

MOMAT is designed as middleware which is developed by Samsung Thales for managing the message of each component. MOMAT provides a data-centric communication based on DDS specification. MOMAT has a powerful user interface and include many options to control message process. All messages and components are controlled by the version such as Subversion and concurrent version system (CVS). CVS is a clientâserver free software revision control system in the field of software development [6].

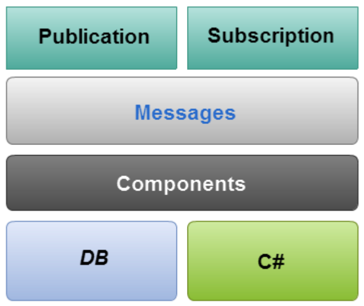


Fig. 1. The architecture of MOMAT

Moreover, it provides a code generation function using interface definition languages (IDL) which is supported by

MDIL library. Using these components (benefits) of MOMAT, message communication and large-scale systems design can be simplified. The MOMAT architecture consisted of four main parts. For operate application C# language was used that integrated with open source relation database management system (RDBMS). Other components, messages and publica-tion/subscription are shown in Fig. 1.

More detail explanation of the MOMAT architecture part as follows:

- $Components$: All of the messages in the MOMAT, can be manage based on components. The component is one of the important item which are divided into 3 layers such as software configuration item (CSCI), computer software components (CSC), and computer software item (CSU). These items gives opportunity to develop documents and manages entire messages during the publish-subscribe process. (Fig. 2). In other words, each component is able to communicate with another component when it participates in global data space to share the data form sensor nodes.
- $Messages$: Each time when user creates a message, it registers to publisher/subscriber. Moreover, it works for user defined data structure for project and topic in DDS.
- $Publication/Subscription$: the publication/subscriber is basic feature in MOMAT. Using this feature we can communicate with each other using global data space (Fig. 2).

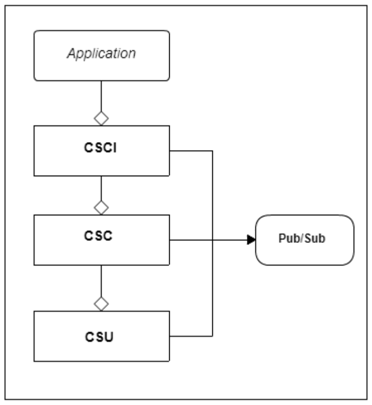


Fig. 2. Pub/Sub process using CSU, CSC and CSCI

B. Data communication of MOMAT

MOMAT topology is designed as client-server and peer to peer architecture using distributed real-time data-centric communication (Fig. 3). The architecture of MOMAT has three main application forms which are shown in Fig. 4. The initial process of aceess from login application is shown in Fig. 4. The all information in application obtains from database periodically using continous query procedure (Fig. 5). In MOMAT each information uploaded to the list in small amount is for enhancing the performance of each application. The important feature of MOMAT is to implement conti-nous query between application and database storage system. However, when hundreds of network nodes need to connect to central database server to data retrieval, the performance of application is significantly decrease and cause to server overload.

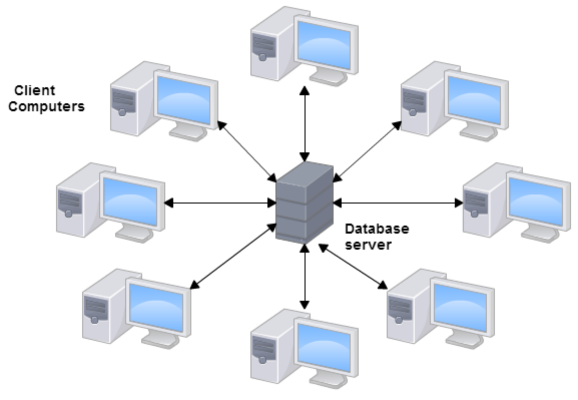


Fig. 3. Network model of MOMAT

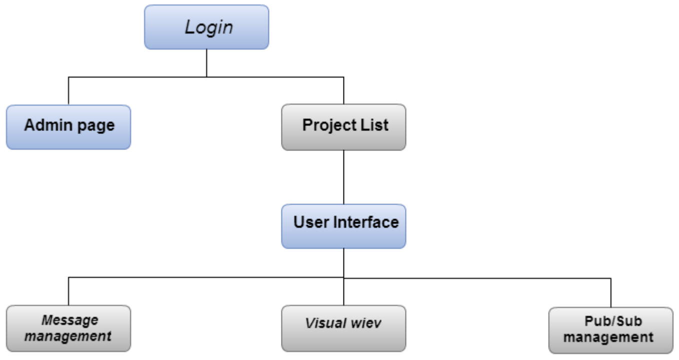


Fig. 4. The MOMAT view configuraion

To solve these problems queriesâsplit management scheme is proposed by grouping multiple continuous queries using group and split mechanism.

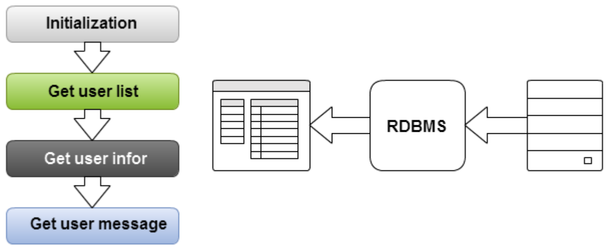
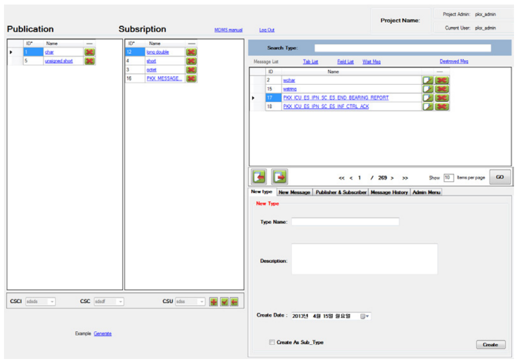


Fig. 5. Initialization procedure of MOMAT

C. Implementation design

The MOMAT is developed by C# language to provide scalability. Fig. 6 shows task screen of MOMAT. As we can see from the Fig. 6 pub/sub items and components are located on the left side of the application. On the right side of the task screen is attached functions as new message, user define data type, topic or modify the message, check history message as well as delete the message.



Project Admin: pkx_admin
Current User: pkx_admin

ID*	Name
1	char
5	unsigned short

ID*	Name
12	long double
4	short
3	octet
16	PKX_MESSAGE...

ID	Name
2	wchar
15	wstring
17	PKX_ICU_ES_IPN_SC_ES_END_BEARING_REPORT
18	PKX_ICU_ES_IPN_SC_ES_INF_CTRL_ACK

CSCI sdsds CSC sdsdf CSU sdss

<< < 1 / 269 > >> Show 10 Items per page

Type Name:

Description:

Create Date : 2013년 4월 15일 월요일

Create As Sub_Type

Fig. 6. Screenshot of task screen

III. Query-split management scheme

A. Related works

The application of distributed system is wide with the rapid development of the technology, such as military operation systems, wireless and wired network and various telecommunication systems. In these systems large numbers of data need to handle continuously update data using continuous query technique (CQ). To provide high performance for such systems several methods was investigated. The grouping method using query split scheme and query group optimization technique was proposed to enhance scalability of the millions of data (queries) [7].

Similar research work has focused on query optimization problem where TelegrapCQ (TCQ) was proposed. TCQ is focused on meeting the challenges that arise in handling large streams of continuous queries over the high-volume, highly-variable data stream.

B. Group optimization using labeling method

Based on query-split scheme strategy, we designed group optimization process using labeling expression. A specific implementation of labeling is shown in Fig. 8. For purposes of demonstration, the MySQL based Data Manipulation Language (DML) is used as an example. The MySQL query in Fig. 8 represent query tree which retrieve information from the database. The submission of the query request gives opportunity to use labeling $(*)$ method. An expression labeling is created for selection predicates by dividing queries into groups.

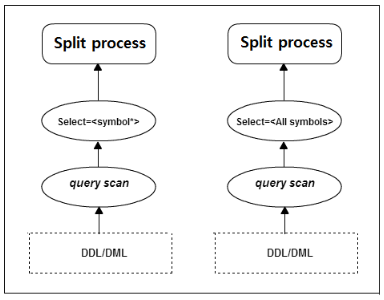


Fig. 7. The split and grouping process

The dividing process of queries is shown in Fig. 7. Lower part in each query represents a DML statement for such information from the database. A new operator ”query scans” is added on the top of DML/DDL process after query parser. The purpose of query scan is to sort between labeled and ordinary queries. The query scans process allows queries with same syntactic structure to be grouped together. Note that for defining between labeling and ordinary query, the labeling process use ”*” $(symbol*)$ in the syntax of the query. This allows users to obtain only last updated attribute (column) value from the database.

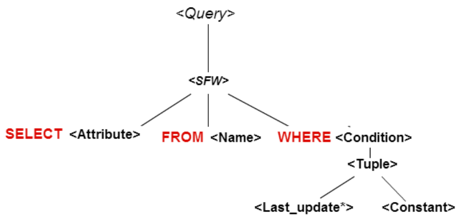


Fig. 8. The labeling of queries

C. Grouping process

The Group plan is the whole process of dividing queries for groups. It is derived from common part of all multi-queries and divided into small group. A group plan allows queries to grouping in different constant [7]. Since the result of the split computation contains results for all the queries in the group, the results must be sent to the server for further processing. The query-split scheme performs filtering by combining a special $Split$ operator with $Grouping$ operator based on constant multi-query value (Fig. 9).

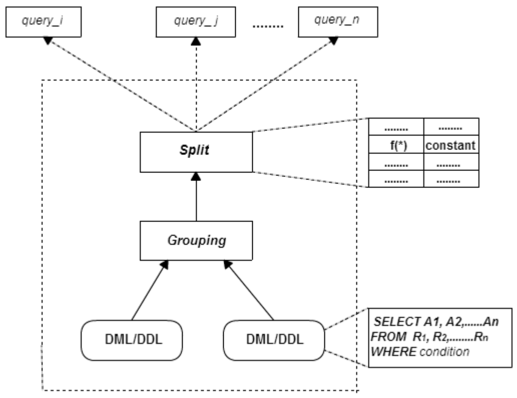


Fig. 9. Query-split scheme using group plan process

The first Grouping operator obtains DML (data manipulation language)/DDL (data definition language) query request and combines it as a group. After group is established by $Grouping$ operator, it forwards to waiting list (buffer), until previous query send by $Split$ operator. The $Split$ operator distributes each query request for small groups to make it easy to send. Moreover, query requests with the same constant value are added to the same query group and share same output stream. Because of $Grouping$ operator combine all query requests in one group and give same destination address, the $Split$ operator set up a new name of destination address and distributes it to each query request.

In general, the number of active query groups should not be increased more than the threshold. Because, in case of overload, the system is not able to handle the big number of query request. Therefore, all small groups are forwarded to buffer and all of them stored in sequential order to control their limitation.

D. Distribution process

In MOMAT's network architecture, each node has some connection directly to the storage node (SN) and several neighbor nodes. For example when the node A searches for certain information, it sends a query message (request) to its storage node (SN). The SN is the main node of the system where all data are stored in storage tables. In case of when system launches split operator, it splits the query request for small query groups and distributes it to the nodes A and B (Fig. 10). Each query group request contains keywords of searched information. The node D that receives the query message from its neighbor node, search it from its local storage system. After message result is obtained by node D, it is forwarded directly to node A for the final process. This approach achieves more effective search performance compared to previous search information in $MOMAT$ system.

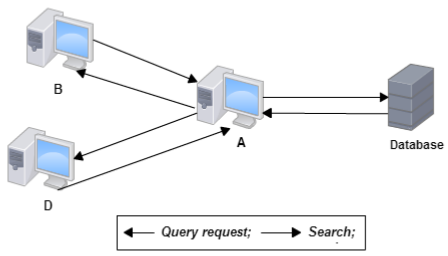


Fig. 10. An example of query routing process.

E. Algorithm

Algorithm 1 represents the query-split management algorithm to increase the scalability of the distributed system. From initial process, the system sets up query sequence $(Q1, Q2....Qn)$ before the main application starts. For example in the algorithm, we consider a query request $Q1$ which is scheduled independently. After the new query request is established, $Grouping$ process is launched by the system. The purpose of $Grouping$ process collects all queries as sequence and make a group to save it in the buffer. During the grouping process each query are (*) labeled by query scan algorithm (Fig. 9) to enhance performance of continuous query processing. After query scan process is done, the split query function slices the obtained information for small optimal groups $<Q_1$= WHERE $sybol(*)$= last-update, Q_2= WHERE $sybol(*)$= last-update;>, $<Q_1$= $C_{value(1)}$, Q_2= $C_{value(2)}$;> and distribute it.

Algorithm 1 Query-split management algorithm

```
1: Initialize($Q_1$, $Q_2 ... Q_n$,)
2: Start application
3: $Q_1$=SELECT * FROM $A_1$,$A_2$,$A_3 ... A_n$ WHERE $symbol$
4: Grouping process $N_{group}$=$Q_1, Q_2 ... Q_n$
5: Lunch query scan process
6: Labeling process $(*)$
7: Split query as group
8: $Q_{group(1)}$=< $Q_1$= WHERE $sybol(*)$= last-update, $Q_2$=
   WHERE $sybol(*)$= last-update>;
9: $Q_{group(2)}$=< $Q_1$= $C_{value(1)}$, $Q_2$= $C_{value(2)}$>;
10: Send query
11: close
```

IV. IMPLEMENTATION AND PERFORMANCE ANALYSIS

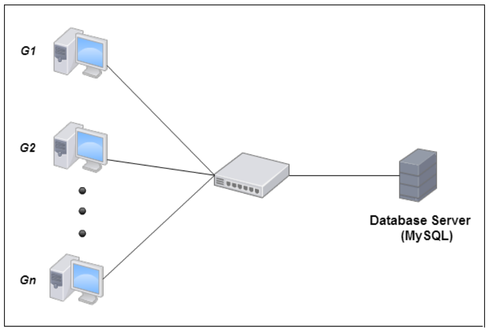


Fig. 11. An Experimental testbed.

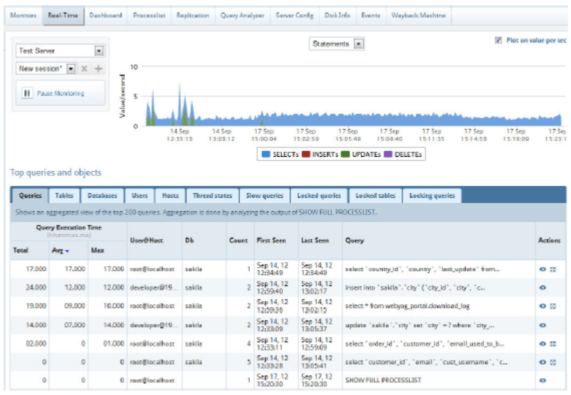
We have performed some experiments in order to develop a performance model for database server. The experiments were performed in our testbed, consisting some number of computers operating as a client and one MySQL database server. An illustration of the testbed is shown in Fig. 11. The computers used in the experiments are ordinary desktop computers with some open source software installed. The computer hardware is a Samsung HD 502 HJ. It is equipped with a 3.10 GHz i3 Intel Celeron processor, 4 GB main memory and Gigabit Ethernet network Interface. The Windows XP is running on the ordinary computers. One computer has a MySQL server installed.

Table 1. Simulation parameters

Parameters	Value
CPU	Intel Celeron i3 3.10GHz
Operation system	Windows XP
Installed memory (RAM)	1 sec
Experiments run time	1-2 hours
Data rate per Stream	200 [tuples/sec]

A. MySQL database monitor and Adviser

For monitor the performance of MySQL database server has been used special software called "MONyog (for Windows)" (Fig. 12). The MONyog software helps MySQL DBAs to manage more MySQL servers, tune their current MySQL servers and find and fix problems with their MySQL database applications before they can become serious problems of costly outages. Moreover MONyog software effectively monitors database environments and provide expert advice on how to optimize performance of database and reduce number of unnecessary connection between tables. The MONyog can continuously monitor queries in real-time and send notifications for queries that take more than a specified amount of time to execute.



Shows an aggregated view of the top 200 queries. Aggregation is done by analyzing the output of SHOW FULL PROCESSLIST.

Query Execution Time (hh:mm:ss.ms)			User@Host	Db	Count	First Seen	Last Seen	Query	Actions
Total	Avg	Max							
17.000	17.000	17.000	root@localhost	sakila	1	Sep 14, 12 12:34:49	Sep 14, 12 12:34:49	select `country_id`, `country`, `last_update` from...	
24.000	12.000	12.000	developer@19...	sakila	2	Sep 14, 12 12:59:40	Sep 14, 12 13:02:17	insert into `sakila`.`city` (`city_id`, `city`, `c...	
19.000	09.000	10.000	root@localhost	sakila	2	Sep 14, 12 12:59:36	Sep 14, 12 13:02:15	select * from webyog_portal.download_log	
14.000	07.000	14.000	developer@19...	sakila	2	Sep 14, 12 12:33:09	Sep 14, 12 13:05:37	update `sakila`.`city` set `city` = ? where `city_...	
02.000	0	01.000	root@localhost	sakila	4	Sep 14, 12 12:33:11	Sep 14, 12 12:59:09	select `order_id`, `customer_id`, `email_used_to_b...	
0	0	0	root@localhost	sakila	5	Sep 14, 12 12:33:28	Sep 14, 12 13:05:41	select `customer_id`, `email`, `cust_username`, `c...	
0	0	0	root@localhost		1	Sep 17, 12 15:20:30	Sep 17, 12 15:20:30	SHOW FULL PROCESSLIST	

Fig. 12. MONyog software.

B. *Implementation result*

Before the query request process started, all connection to the server was set up and MySQL monitoring software was initialized to ready to go. The result of information send by MySQL database system using general MOMAT approach is given in Fig. 13.

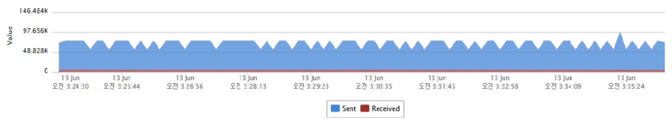


Fig. 13. Amount of information is send by MySQL using general approach

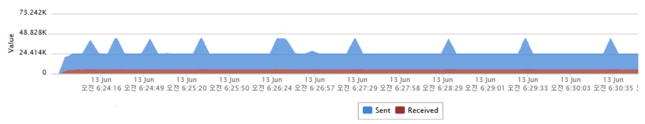
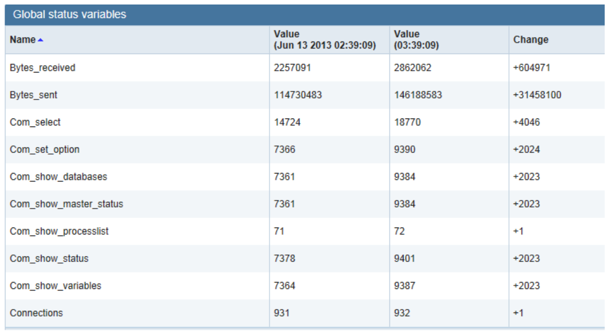


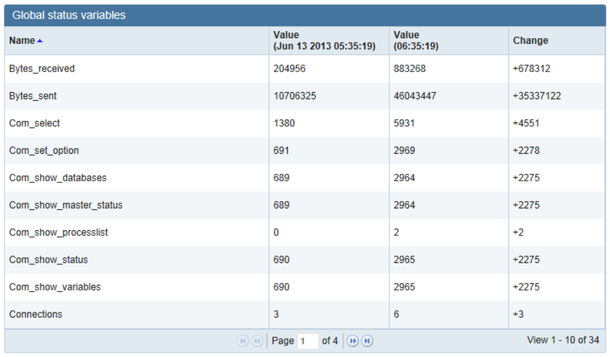
Fig. 14. Amount of information is send by MySQL using query-split management scheme approach

Based on Figures 13 and 15 we can predict that if each application request 2000 tuples in one time the amount of information send will be huge and workload of central database system significantly increase. Furthermore, the CPU utilization in server node compering with normal system is higher. In case of proposed approach (Fig. 14), server node utilizes less resource to distribute information to the nodes compering with Fig. 13.


Global status variables

Name	Value (Jun 13 2013 02:39:09)	Value (03:39:09)	Change
Bytes_received	2257091	2862062	+604971
Bytes_sent	114730483	146188583	+31458100
Com_select	14724	18770	+4046
Com_set_option	7366	9390	+2024
Com_show_databases	7361	9384	+2023
Com_show_master_status	7361	9384	+2023
Com_show_processlist	71	72	+1
Com_show_status	7378	9401	+2023
Com_show_variables	7364	9387	+2023
Connections	931	932	+1

Fig. 15. Total amount of information (bytes) send by nodes using general approach


Global status variables

Name	Value (Jun 13 2013 05:35:19)	Value (06:35:19)	Change
Bytes_received	204956	883268	+678312
Bytes_sent	10706325	46043447	+35337122
Com_select	1380	5931	+4551
Com_set_option	691	2969	+2278
Com_show_databases	689	2964	+2275
Com_show_master_status	689	2964	+2275
Com_show_processlist	0	2	+2
Com_show_status	690	2965	+2275
Com_show_variables	690	2965	+2275
Connections	3	6	+3

Page 1 of 4 View 1 - 10 of 34

Fig. 16. Total amount of information (bytes) send by nodes using query-split management scheme

Figures 15 and 16 show total amount of information (bytes) when data retrieval is performed by each application. Compering with Fig. 15 which uses general approach, Fig. 16 utilizes less resource to obtain information form database system.

V. CONCLUSION AND FUTURE WORK

The purpose of this paper is to develop a scalable object-oriented middleware continuous query system using a labeling method to split the query into small groups. The previous version of MOMAT, MDMS (Message Definition and Management System) [8] considers only a limited number of modeling tools for developers to design the naval combat system. Furthermore, MOMAT is not able reach high scalable in case of big number of queries. To solve this issue, we propose "query-split" methodology that split the input queries into small groups to make the system more scalable. We also implemented new powerful user interface to MOMAT which has many options to model publish-subscribes message process.

For future work, we plan to concern about QoS feature in MOMAT. In addition, this paper plans to measure workload of each application considering the application execution time and maintenance.

ACKNOWLEDGMENTS

This research was financially supported by the Ministry of Education, Science Technology (MEST) and National

Research Foundation of Korea (NRF) through the Human Resource Training Project for Regional Innovation 2013.

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (NO. 2011-0025409).

References

[1] D.-S. Kim, Y. S. Lee, and H. S. Park, “Maximum allowable delay bounds of networked control systems,” *Control Engineering Practice, Elsevier*, vol. 11, no. 11, pp. 1301–1313, 2003.

[2] H. Arciszewski, T. de Greef, and J. van Delft, “Adaptive automation in a naval combat management system,” *IEEE Transactions on Systems, Man and Cybernetics*, vol. 39, no. 6, pp. 1188–1199, 2009.

[3] D.-S. Kim and S. Lee, “Feasibility analysis of hybrid control networks based on common industrial protocol,” *Computer Standards & Interfaces, Elsevier*, vol. 33, no. 4, pp. 357–366, 2011.

[4] W. R. Otte, A. Gokhale, D. C. Schmidt, and J. Willemsen, “Infrastructure for component-based dds application development,” in *Proceedings of the 10th ACM International Conference on Generative Programming and Component Engineering*, 2011, pp. 53–62.

[5] W. Kang and S. H. Son, “Data services in distributed real-time embedded systems,” 2008.

[6] “Concurrent versions system,” http://en.wikipedia.org/wiki/Concurrent_Versions_System.

[7] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang, “Niagaracq: A scalable continuous query system for internet databases,” in *SIGMOD Conference*, 2000, pp. 379–390.

[8] J. Y. Yu and J. Park, “A massage management for cooperative message-based interface development,” in *Journal of Korean Insitute of Information Scientists and Engineers*, 2008, pp. 639–613.