분산 시스템의 적응형 내결합성 및 QoS 미들웨어 지원

조바니 카가라반*, 김석수*
*한남대학교 멀티미디어학과
e-mail:gcagalaban@yahoo.com, sskim@hnu.ac.kr

An Adaptive Fault Tolerant and QoS-Enabled Middleware Support in Distributed Systems

Giovanni A. Cagalaban*, Seoksoo Kim*
*Dept. of Multimedia, Hannam University

요 약

Normally, a distributed computing environment is flexible in controlling complex embedded systems but their software components are becoming complex as these systems are equipped with several platforms and attached to various electronic devices, sensors, and actuators. These systems requires inter-object communication mechanisms to provide fault tolerant and QoS-enabled middleware service support in a distributed system. Generally, a middleware performs analysis of the parameters to ensure the availability and reliability of data dissemination. This paper focuses in particular to designing an application middleware for the specific scenario to improve the high availability and fault tolerance of data thus improving the QoS (Quality of Service) of a distributed system. The performance of an adaptive and highly reliable middleware can be significant based on the selection of vital parameters of the system.

1. Introduction

Distributed computing is becoming increasingly widespread and important where distributed real-time embedded (DRE) systems have potentially powerful approach for accessing large amounts of computational power [16]. These systems include telecommunication networks, telemedicine, manufacturing process automation, and defense applications [17]. Distributed computing deals with computational decentralization across a number of processors which may be physically located in different components, subsystems, systems, or facilities [12]. A distributed system must provide various mechanisms for process synchronization and communication, for dealing with the deadlock problem, and for dealing with a variety of failures that are not encountered in a centralized system [14].

Middleware is a set of common business-aware services that enable applications and end-users to interact with each other across a network. Essentially, middleware is a software that resides above the network and below the business-aware application software [15]. Developing adaptive programming paradigm in a distributed system, a middleware generally consists of a mechanism to allow methods to be invoked on remote objects, plus services to support the naming and location of objects in a system.

Traditionally, middleware systems are designed for a particular application domain or deployment scenario. As middleware systems become distributed and support mission critical application systems, reliability of these systems becomes a critical issue. If there is a point of failure in a distributed system, the services offered is still available from other component of the middleware. Present research has focused on investigating the possibility of enabling middleware to serve multiple domains and deployment environments.

In recent years, systems have emerged that support reconfigurability, allowing systems to be

customized for a specific task. An adaptive fault-tolerant system has the ability to change behavior and functionality. Its functional behavior can be modified dynamically to optimize for a change in environmental conditions or requirements[10]. These adaptations can be triggered by changes made to a configuration file by an administrator, by instructions from another program, or by requests from its users. The primary requirements of these systems are measurement, reporting, control, feedback and stability [13].

This paper proposed a new framework to improve quality of service in distributed systems. In particular, this paper proposes a framework for a scalable data dissemination using multicast communication. The framework integrates the adaptability and fault-tolerance of the middleware and scalable date dissemination of data for the availability and high performance of the system.

2. Related Work

An early research work on scalable data dissemination is the teletext system on television in Western Europe in early 80s [7]. It uses the spare bandwidth allocated in the European television channels to implement multicast push data dissemination.

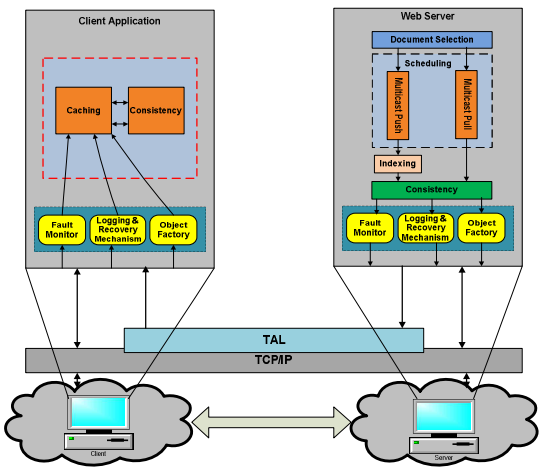
In data networks, works on multicast push include high-throughput multiprocessors database systems over high-bandwidth networks [5]. A middleware technology is developed which aims to realize this promised benefit of distributed systems to support the distributed application programmer by offering powerful high availability abstractions, enabling the programmer to concentrate on the application [11].

A semantic middleware architecture is designed to detect errors, analyze causes of errors, and plan semantically meaningful strategies to deal with a problem with associating fault and service ontology in UbiComp environment and they implemented a referenced prototype, Web-service based Application Execution Environment (Wapee), as a proof-of-concept, and showed the efficiency in runtime recovery [9]. The DBIS-Toolkit [3] introduces a gateway for data delivery. The toolkit provides a set of components to adapt various data dissemination methods with each other.

Additionally, there are numerous Internet multicast protocols that have been proposed recently. The original focus was on extension to the IP protocol to enable multicast within the core of the Internet infrastructure[a8]. But the focus has shifted toward multicast schemes implementing functionality using application-layer modules that execute at the network edges due to few providers that have allowed IP multicast, This middleware provides extension at the transport layer by utilizing advanced data management functionality.

3. Proposed Middleware Framework

The outline of the proposed framework of the application middleware is shown in Figure 1. The configuration demonstrates the major components that is linked in a multicast environment. The middleware deals with the following data management function in multicast data dissemination such as document selection, scheduling, consistency, caching and indexing. The middleware framework is designed from the individual components that can be selected based on the underlying multicast transport mechanism.



[Fig. 1] Middleware framework for data dissemination

The application is a highly scalable Web server and the underlying transport is IP multicast . This is based on the work of [2]. With this, the Web server can disseminate data by employing any of the following three schemes: multicast push, multicast pull and unicast pull. Typically, if an application uses all three data dissemination schemes and calls the middleware, the following sequence of modules occurs. Statistics on application data units (ADUs) are gathered at regular intervals. When the statistics have been collected, the Web server classifies ADUs into hot, warm, and cold documents to be disseminated using multicast push, multicast pull and unicast pull schemes.

The goal of the Web server application is to scale to a large client population and scalability is accomplished by using the data dissemination middleware. Since the Web server broadcasts an index of sorted URIs on the multicast push channel to, the client is capable of determining in a sequential order whether a needed ADU is in the set of current hot broadcast. First, If the ADU is not in the set of hot broadcast, the Web client makes a specified request to the Web server and immediately starts to listen to the multicast pull channel. Next, if the ADU is in the set of cold broadcast, the requested ADU is returned on the unicast pull connection. Otherwise, the Web client waits on the multicast pull channel until the requested ADU is transmitted.

In multicast push scheduling, the Web server regularly broadcast hot resources to the Web clients. The frequency and order in which pages are bradcast is determined by the multicast push scheduling component. Pages are then injected at a specified rte that is statistically determined from measurements of network characteristics. In this component, we implemented the MAD algorithm[4]. The algorithm is proven to achieve 2-approximation factor and is easy to be implemented.

For the multicast pull scheduling, it was implemented using the Longest Total Stretch First (LTSF) algorithm [1]. The multicast pull scheduling component resolves contention among client requests for the use of the warm multicast channel. It also establishes the order in which pages are sent over the channel.

The implementation of client cache modules take into account the consistency and replacement so it can significantly improve performance by reducing perceived latency of desired pages. In this component, the provably optimal caching algorithm by [8] is implemented to provide client-side caching of ADUs.

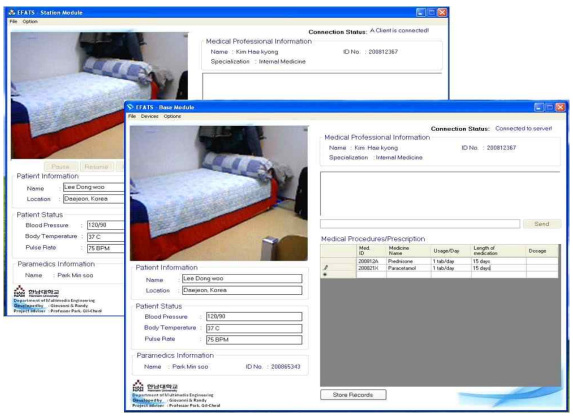
The Transport Adaptation Layer (TAL)is a thin layer that does not implement features that are inappropriate for the underlying transport. It renders middleware and applications independent of the particular choice of a multicast transport protocol.

4. Implementation

The developed middleware can support the gathering and monitoring of the data needed for the prompt delivery of first aid treatment to pre-hospital patients and also for the up-to-date dissemination of critical information to a number of emergency medical service personnel for diagnosing and treatment. An emergency first aid treatment system is developed to gather and monitor the transmission of patient's vital signs from the incident cite or the moving ambulance to the hospital and perform remote medical procedures or examinations. This system provide an expert-based healthcare in emergency cases with the aid of modern communication technologies such as wireless sensors and cameras.

The system utilizes the middleware to disseminate the data being gathered. The server-side database of the system was created in Microsoft SQL Server. It stores the relevant patient information such as temperature, heart rate, blood pressure, breathing rate and other physiological characteristics of the patient taken

from sensors and cameras. The main application of the system is written in Microsoft Visual C# which interacts with the client-side modules which is also developed in Microsoft Visual C#. The main graphical user interfaces of server-side and client-side modules are shown in Figure 2. The server queries the database for the relevant patient information and it is processed by the middleware. The middleware then sends these vital information in XML format on the appropriate multicast channel.



[Fig. 2] Snapshot of the program interface

When a patient is being monitored, his relevant information is requested using the middleware. If the requested data is not locally cached, then it is fetched from the server. These information are then gathered and returned by the middleware to be displayed on the client-side interface.

5. Conclusion

Building a distributed system in a heterogeneous is difficult and expensive due to exchange of complex data, different encoding of data types, synchronization and real parallelism, and the need for atomic sequence operations. As such, there is a need for middleware to be deployed in a distributed system environment.

In this paper, we presented a middleware that unifies and extends support for data management in multicast data dissemination. It emphasized on the improvement of the performance of the middleware with the adoption of the efficient algorithms.

References

[1] S. Acharya and S. Muthukrishnan, "Scheduling on-demand broadcasts: New metrics and algorithms." *Proceedings of theFourth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, 1998.

[2] K. Almeroth, M. Ammar and Z. Fei, "Scalable Delivery of web Pages Using Cyclic Best-Effort (UDP) Multicast." *Proceedings of the Seventeenth Annual joint Conference of the IEEE Computer and Communications Societies*. 1998.

[3] M. Altinel, D. Aksoy, T. Baby, M. Franklin, W. Shapiro, and S. Zdonik. "Dbis-toolkit: Adaptable middleware for large scale data delivery." *Proceedings of the ACM SIGMOD Conf*erence, pp.544 - 546, 1999.

[4] A. Bar-Noy, R. Bhatia, J. Naor and B. Schieber, "Minimizing service and operation costs of periodic scheduling." *Mathematical Operations Resesearch*. 2002.

[5] T. Bowen, G. Gopal, G. Herman, T. Hickey, K. Lee, W. Mansfield, J. Raitz and A. Weinrib. "The Datacycle Architecture" *Communications of the ACM*. 1992.

[6] S. Deering. "Multicast routing in internetworks and extended LANs." Proc. of the ACM SIGCOMM Conf.", pp.55 - 64, 1988.

[7] D. Gifford, "Polychannel Systems for Mass Digital Communications." *Communications of the ACM*. 1990.

[8] S. Khanna and V. Liberatore, "On broadcast disk paging." *SIAM Journal on Computing*, 2000.

[9] Y. Kim, E. Kim, B. Jeon, I. Ko, and S. Park, "Wapee: A Fault-Tolerant Semantic Middleware in Ubiquitous Computing environments", *IFIP International Federation for Information Processing*, 2006.

[10] J. Loyall, et.al. "Comparing and Contrasting

Adaptive Middleware Support in Wide-Area and Embedded Distributed Object Applications. *Proceedings of the 21st International Conference on Distributed Computing Systems*, 2001.

[11] P. Murray, R. Fleming, P. Harry and P. Vickers, "Somersault: Enabling Fault-Tolerant Distributed Software Systems", 1998

[12] B. Palmintier, C. Kitts, P. Stang, M. Swartwout, "A Distributed Computing Architecture for Small Satellite and Multi-Spacecraft Missions," *16th Annual AIAA/USE Conference on Small Satellites*, August 12 - 15, 2002.

[13] R. Schantz and D. Schmidt, "Middleware for Distributed Systems: Evolving the Common Structure for Network-centric Applications. *Encyclopedia of Software Engineering*. pp 801-813. 2001.

[14] A. Silberschatz, P. B. Galvin, G. Gagne "Operating System Concepts", 7th Edition, 2006.

[15] A. Uma, "Object-Oriented Client / Server Internet Environments - The IT Infrastructure", 1999.

[16] C. Woods, J. Frey and J. Essex, "The Application of Distributed Computing to the Investigation of Protein Conformational Change", 2004.

[17] J. Zhang, L. DiPippo, "A Real-Time Distributed Scheduling Service for CORBA Systems" *IEEE Transactions On Real Time Computing*, vol. 32, pp.244-250, Feb. 2007.