An NMS Architecture for Large-scale Networks*

Byungjoon Lee[1], Youhyeon Jeong[1], Hoyoung Song[1], and Youngseok Lee[2]

[1] Electronics and Telecommunications Research Institute, Republic of Korea
{bjlee, yhjeong, hsong}@etri.re.kr
[2] Chungnam National University, Republic of Korea
lee@cnu.ac.kr

Abstract. One of the key characteristics of the contemporary network is its scale. The cloud data center network, which is considered as a key component of the current Internet, is normally composed of tens of thousands of devices. Besides, networks which connect data centers and their users are getting larger and becoming more complex because of the growing bandwidth requirement and user demands. This paper suggests a novel network management software architecture, which is suitable to manage such large-scale networks in a scalable and highly available manner. The software architecture has been designed based on consistent hashing. In this paper, we provide a formal definition of the software architecture, and evaluation results to show its feasibility.

Keywords: NMS, Consistent Hashing, CORD

1 Introduction

One of the key characteristics of the contemporary network is its scale. Normally, a cloud data center network is composed of more than 50,000 elements [1]. The cloud data centers are connected with each other and end-users through high-performance transport networks, which are also getting larger and becoming more complex. The scale of such networks casts several important questions: (1) *does the architecture provide a performance improvement method according to the size of managed networks?* (2) *is it possible to quickly recover an NMS from software component failures?*

The management system software for such networks should be able to monitor thousands of network elements within a reasonable time bound and to configure the huge number of network elements simultaneously to maintain their configurations consistently [2][3]. It means that an NMS should be horizontally scalable; we should be able to enhance its performance by adding more servers, and we should be able to predict the performance gain. Otherwise, it is impossible to calculate the cost to provide reasonable network management performance.

* This research is supported by the IT R&D program of KCC/KCA (11911-05003: R&D on Smart Node Technology for Cloud Networking and Contents Centric Networking).

Besides, an NMS for such networks should provide high availability which guarantees the fast failover from software failure. The unwanted downtime of an NMS might lead to the loss of revenue of the network service providers.

In this paper, we suggest an NMS software architecture that focuses on the *high scalability* and *high availability*. The original version of the architecture was first publicized in [4]. This paper is an extension to the original version, with more formal definitions and more evaluation results. We implemented the software architecture as a middleware, called *COre library for Rapid Development of NMSs (CORD)*.

This paper is organized as follows. Section 2 summarizes previous important studies. In Section 3, we present the CORD architecture in detail. Section 4 presents the performance evaluation results of CORD. Finally, concluding remarks and future research directions are presented in Section 5.

2 Related Works

A multi-tier architecture is a very popular software architecture that enhances overall manageability and scalability by splitting a software into multiple layers. Each layer consists of a similar set of functions following different development strategies from the other layers. The architecture has been adopted to network management systems including [5]. However, the multi-tiered architecture does not allow linear performance enhancement by adding more servers to the NMS server pool; the whole system should be stopped to reconfigure the connections between layers and servers, and there's no guarantee that load is evenly distributed and automatically reconfigured.

CORBA-based network management systems [6][7][8][9] model software components and network elements as objects, and have them cooperate to manage networks. Those systems solve many problems of the traditional multi-tier management architecture. CORBA-based network management systems are scalable to the size of the managed networks because it is possible to distribute NMS software components on multiple nodes and make them share management workloads. But the scalability of the CORBA-based management systems is limited; though developers are able to deploy objects across multiple servers, it is not possible to predict the volume of management workloads assigned to each object because CORBA does not support load-balancing on objects.

P2P-based network management systems [10][11][12][13] mainly focus on attaining network scalability by allowing a management overlay network is automatically configured. However, above studies did not pay attention to the scalability or availability of a network management system itself; the scalability and availability of an NMS that sits on top of the P2P network does not change by adding more P2P nodes. Though there was a study which delegated some of the management functionalities to P2P peers [14], they did not study how such delegation might enhance the overall scalability and availability of a system.

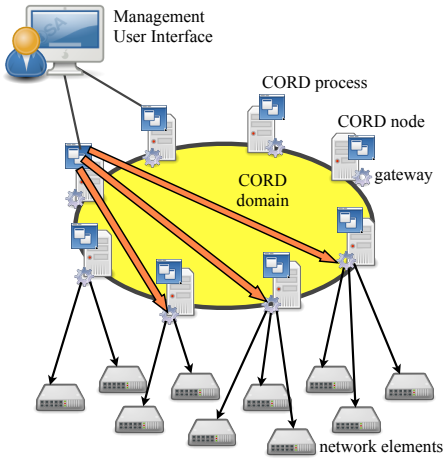

Fig. 1. CORD network management architecture

3 CORD: An Alternative NMS Software Architecture

In designing an alternative NMS software architecture and implementing it as a middleware, CORD, we have used the consistent hashing as our tool [4]. In this paper, we achieve predictable scalability and high availability by using consistent hashing (1) to distribute management workload evenly to all NMS servers, and (2) to design a fast software component failover mechanism.

Fig. 1 is a conceptual diagram of one CORD system. One CORD-based NMS system that manages one network domain is called a *CORD domain*, which is a set of CORD nodes and CORD processes. *CORD node* is where the CORD middleware is installed, and *CORD process* is a software component process which is the part of an NMS. CORD processes run on CORD nodes. For every CORD process, we assign an unique identifier. CORD nodes communicate with each other by *CORD protocol*.

If a network manager sends some network management policies to a specific CORD process using a user interface, the process identifies a set of *management commands* for applying the policies to the network. After the identification, the process evenly distributes the management commands to all CORD nodes. Every CORD node that receives a command delivers it to a network element.

Consistent hash function of CORD We denote the consistent hash function of CORD as $\mathcal{F}$. $\mathcal{F}(x)$ returns the IP address of a CORD node that manages the key x. We use $\mathcal{F}(x)$ to determine a CORD node to which we register the location of a CORD process x. We also use $\mathcal{F}(x)$ to calculate which CORD node manages a network element x. In implementing $\mathcal{F}$, we assumed that every CORD node knows all IP addresses of all CORD nodes in the same domain. To satisfy the assumption, we use a simple discovery protocol [4].

Thus, $\mathcal{F}(k)$ is defined as $\mathcal{F}(k) = argmin_{\forall s_i \in S}\mathcal{D}(k, s_i)$ where S is a set of all IP addresses of all CORD nodes. $\mathcal{D}(x, y)$ is the distance between $\mathcal{H}(x)$ and $\mathcal{H}(y)$ on a circular hash space $\mathcal{H}$, measured in a counter-clockwise direction. Thus, $\mathcal{F}(k)$ maps k to the IP address of the k's counter-clockwise adjacent node on $\mathcal{H}$.

Process registration and Communication Let there be several processes started at the node s_k, and the set of their identifiers be $P_k = \{p_{k,1}, \ldots, p_{k,n}\}$. The node s_k records the fact that process $p_{k,i}$ is started at the node s_k by sending a *registration message* to the node $\mathcal{F}(p_{k,i})$ (*resolver* node of $p_{k,i}$).

For example, let there be three CORD processes with the same identifier value k, which are being started on three different CORD nodes IP_1, IP_2, and IP_3. They send registration messages to the same resolver $\mathcal{F}(k)$. After receiving the messages, the resolver node builds a location table shown in Fig. 2. The resolver sends an *activation message* to the first node that sent the registration message, and replies *stand-by messages* to the other nodes. A node that receives the activation message changes the status of the process k into 'activated'. The other nodes would turn the processes into 'stand-by' status. The registration procedure is executed whenever there is a change on $\mathcal{F}(p), \forall p \in P_k$.

A process $p_{m,n}$ that wants to communicate with process $p_{i,j}$ first *queries* the resolver $\mathcal{F}(p_{i,j})$ where $p_{i,j}$ is. The resolver replies the IP address of the active $p_{i,j}$. Using the IP address, $p_{m,n}$ is able to communicate with $p_{i,j}$. Because this query-based communication scheme does not require a priori knowledge of IP addresses, it is possible to move processes to different locations transparently.

Failover: process and system faults We have implemented a fast process failover mechanism using the resolver nodes. For example, if the active process k in Fig. 2 fails, the CORD node IP_1 detects the failure, and sends a *deregistration messsage* to its resolver. The resolver removes entry $< k, IP_1 >$ from the location table, and sends an activation message for the next registered location of process k (in this case, IP_2). The node IP_2 immediately activates the process k.

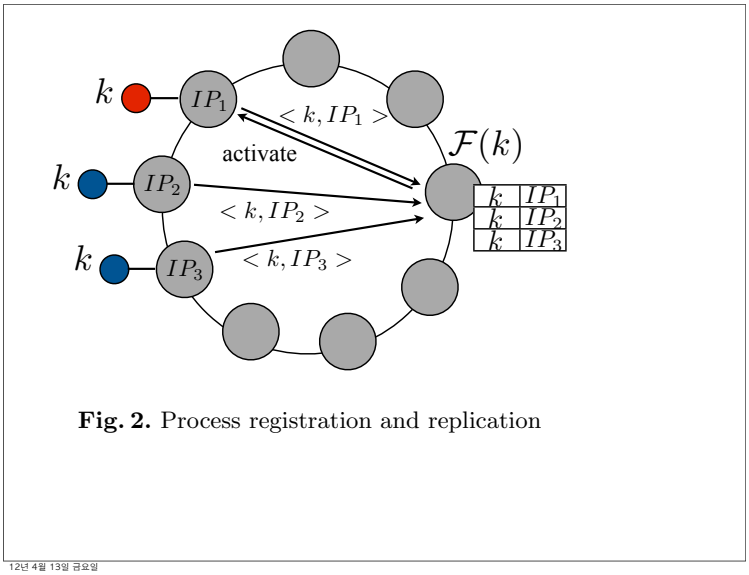

Fig. 2. Process registration and replication

12년 4월 13일 금요일

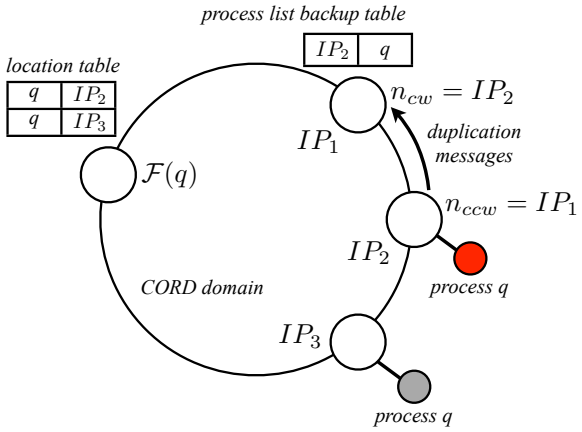

Fig. 3. Process list backup table

The process failover time is as follows:

$$d_{pf} + s_d + s_a \approx d_{pf} + 2 \cdot s \tag{1}$$

d_{pf} is time for a node to detect the failure of a process; s_d is time for the node to deliver a de-registration message to resolver; and s_a is time for the resolver to deliver an activation message. Thus, the process failover time is determined by the network delay s, which is very small in the contemporary networks.

Basically, CORD handles system failures by re-registration; if one CORD node fails, $\mathcal{F}$ of all CORD nodes are changed, and the change forces every node to re-register their process locations. In result, a new active processes is elected.

However, the time complexity of this basic failover scheme is $O(N)$ where N is the number of nodes in the domain, because it takes about $\frac{N}{2} \cdot s$ for an *address removal message* to be delivered to all CORD nodes. CORD solves this issue using *process list backup table*. That is, every node sends its 'active' process identifiers to its counter-clockwise neighbor n_{ccw} using *process list replication messages* (Fig. 3). n_{ccw} saves the process identifier into the process list backup table. The backup table entries are not used until the loss of n_{cw} is detected.

On detecting the loss of IP_2, IP_1 sends *address removal* messages for all processes in the backup table to their resolvers. Therefore, $\mathcal{F}(q)$ receives an address removal message from IP_1. Then the resolver $\mathcal{F}(q)$ removes location table entries for IP_2, determines new active process locations, sends process activation messages to the determined locations, and updates $\mathcal{F}$.

Therefore, the system failover time is reduced to:

$$d_{nf} + s_d + s_a \approx d_{nf} + 2 \cdot s \tag{2}$$

where d_{nf} is a time to detect the loss of neighbor node.

Scalability with even load distribution To achieve predictable scalability, CORD evenly distributes management workloads (management commands) to all CORD nodes using $\mathcal{F}$. Thus, we are able to enhance the management performance linearly. Let $D = \{e_1, e_2, \ldots, e_n\}$ be the set of IP addresses of all network elements within a single management domain. A CORD node m manages a set of network elements E such that $\mathcal{F}(e) = m, \forall e \in E, E \subseteq D$. By applying the techniques mentioned in [15][16][17], we tuned that the number of network elements assigned to each CORD node is close to K/N where $K = \|D\|$ and N is CORD domain size.

A process $p_{i,j}$ sends a management command for a network element x to $\mathcal{F}(x)$. The receiver node $\mathcal{F}(x)$ interacts with x on behalf of $p_{i,j}$ and sends a reply to $p_{i,j}$. Thus, by adding more CORD nodes to a CORD domain, we are able to achieve almost-linear performance improvements because the management workload is evenly distributed to all CORD nodes automatically.

4 Performance Evaluation

4.1 Failover recovery time

The process and system failover performance of CORD is provided in [4]. In addition, to verify that the system failover time does not depend on the size of a CORD domain, we collected ping messages between CORD processes that have RTT values greater than 15ms per every domain size. Such longer delay (normally below 1ms) are believed to be introduced by the system failover. For the experiments, we used 128 OpenVZ virtualized CORD nodes. For each domain size, we emulated 256 sequential node failures by picking 256 CORD nodes in sequence and turn them on/off in turn. As depicted in Fig. 4, the RTT values do not vary much by the size of CORD domain, which supports the argument that the system failover time of CORD does not depend on the size of a CORD domain.

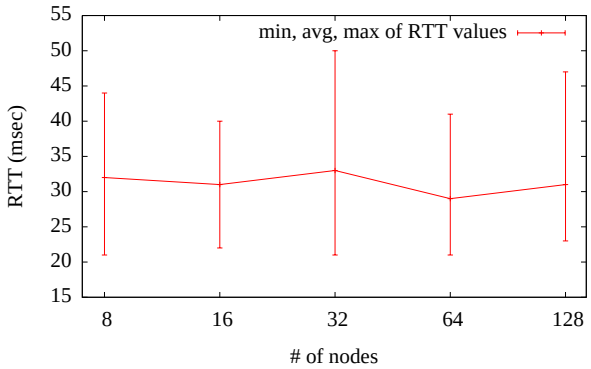

Fig. 4. Average failover time according to CORD domain sizes

It is important to distribute device addresses to multiple CORD nodes evenly to guarantee scalability. To check if our consistent hash function guarantees even distribution, we implemented the consistent hash function $F(x)$ based on the techniques mentioned in [12] and tested it with 65,536 IP addresses and 16 to 128 CORD nodes to see how well the addresses are distributed. Through the experiment, we verified that the standard deviation of the number of addresses assigned to one CORD node was around 5%, which is a marginal value. Therefore, the CORD consistent hash function is able to distribute monitoring workload for the addresses evenly to all CORD nodes.

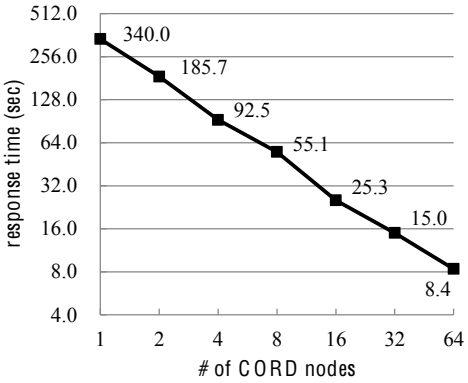

Figure 4. Performance for variable size of a CORD cloud for 20,000 routers: as the number of CORD nodes increases, the response time for 20,000 monitoring method calls (=140,000 SNMP GET operations) is reduced.

4.2 Load distribution and linear performance improvement

Then, we measured how the monitoring performance of a CORD cloud varies by adjusting its size. For experiment, we executed 20,000 monitoring method calls (= 140,000 SNMP GET requests) on 20,000 network elements using one to 64 CORD nodes. Each method call is equivalent to seven SNMP GET requests. Whenever a call is issued, the call is delivered to a CORD node selected by the consistent hash function. The RMP processes incoming method calls sequentially. The network element is chosen sequentially from the pool of 20,000 emulated Cisco, Juniper, and Nortel routers. As shown in Figure 4, we could enhance the monitoring performance by adding commodity CORD servers to the CORD cloud.

To measure how the management performance of a CORD domain is improved by the consistent hash function, we measured a total response time for executing SNMP GET queries for 20,000 network elements by varying the size of the CORD domain. To prepare 20,000 network elements, we used *node emulator* developed by ETRI. Each emulated network element is Cisco, Juniper, or Nortel router. We also prepared a management command which queries seven different SNMP scalar OIDs. Therefore, in our experiment, we perform 140,000 SNMP GET queries. The measurement result given in Fig. 5 suggests that the total response time is half-reduced by increasing the size of the CORD domain twice.

Fig. 5. CORD domain size and management performance

5 Conclusion

The size and complexity of contemporary networks are evolving in an unprecedented way. The advent of cloud computing centers changed the scale of networks, and the huge traffic between them is modifying the transport network structure connecting the centers and users. Thus we need a novel network management software architecture which enables network management systems to easily deal with the scale of the underlying networks in a highly available manner. To meet the demand, we have proposed a new management software architecture, called CORD, which is defined on the consistent hash function.

The scalability of the proposed architecture is predictable because we can enhance the management performance linearly by adding more servers to the management server pool. The architecture provides sub-50ms failover performance which is adequate to support highly-available management systems [19].

CORD has been successfully applied to the implementation of several network management system, including PBB-TE path configuration and management system (PPS). In the near future, we are planning to apply CORD to the inter-cloud traffic management systems.

References

1. A. Greenberg, J. Hamilton, D.A. Maltz, and P. Patel, *The cost of a Cloud: Research Problems in Data Center Networks*, ACM SIGCOMM Computer Communication Review (CCR), Vol. 39, Issue 1, January 2009.
2. S. Kim, M-J. Choi, and J. W. Hong, *Towards Management Requirements of Future Internet*, 11th Asia-Pacific Network Operations and Management Symposium (APNONS) 2008.
3. M-J. Choi and J. W. Hong, *Towards Manageability in the Next Generation Networks*, IEICE Transactions on Communications, Vol. E90-B, No. 11, Nov. 2007, pp.3004-3014.
4. B. Lee, Y. Jeong, H. Song, and Y. Lee, *A Scalable and Highly Available Network Management Architecture*, GLOBECOM 2010.
5. Web NMS 5 framework, http://www.webnms.com/webnms/index.html.
6. P. Haggerty, and K. Seetharaman, *The Benefits of CORBA-based Network Management*, Communications of the ACM, Vol. 41, No. 10, pp. 73-79, October 1998.
7. M. Jeong, J. Kim, J. Kwon, and J. Park, *Design and Implementation of CORBA-based Network Management Applications within TMN Framework*, APNOMS 1999.
8. M. Leppinen, P. Pulkkinen, and A. Rautiainen, *Java and CORBA-based Network Management*, IEEE Computer, Vol. 30, No. 6, pp. 83-87, Jun 1997.
9. J. Pavon, and J. Tomas, *CORBA for Network and Service Management in the TINA Framework*, IEEE Communications Managazine, Vol. 36, No. 3, pp. 72-79, March 1998.
10. L. Z. Granville, D. M. Rosa, A. Panisson, C. Melchiors, M. J. B. Almeida, and L. M. R. Tarouco, *Managing Computer Networks using Peer-to-Peer Technologies*, IEEE Communications Magazine, Vol. 43, No. 10, pp. 62-68, October 2005.
11. R. State, and O. Festor, *A management platform over a peer to peer service infrastructure*, ICT, 2003.
12. C. Simon, R. Szabo, P. Kersch, B. Kovacs, A. Galis, and L. Cheng, *Peer-to-peer management in Ambient Networks*, 14th IST Mobile and Wireless Communications summit, Dresden, Germany, June 19-23, 2005.
13. A. Fiorese, P. Simoes, and F. Boavida, *A P2P-based Approach to Cross-domain Network and Service Management*, AIMS, 2009.
14. R. Carroll, C. Fahy, E. Lehtihet, S. Meer, N. Georgalas, and D. Cleary, *Applying the P2P paradigm to management of large-scale distributed networks using a Model driven approach*, NOMS, 2006.
15. D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, *Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web*, in Proc. ACM Symposium on Theory of Computing, 1997.
16. D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, L. Matkins, and Y. Yerushalmi, *Web Caching with Consistent Hashing*, WWW, 1999.
17. I. Stoica, R. Morris, D. Karger, M. Frans Kaashoek, and H. Balakrishnan, *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*, ACM SIGCOMM, 2001.
18. G. DeCandia, D. Hastrorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, *Dynamo: Amazon's Highly Available Key-value Store*, SOSP, 2007.
19. O. Bonaventure, C. Filsfils, and P. Francois, *Achieving sub-50 milliseconds recovery upon BGP peering link failures*, IEEE/ACM Transactions on Networking, Vol. 15, Issue 5, pp. 1123-1135, October 2007.