# Scale-out Architecture for Service Order Processing Systems

Masafumi Shimizu, Hikotoshi Nakazato, and Hikaru Seshake
NTT Network Service Systems Laboratories,
NTT Corporation
3-9-11 Midori-cho, Musashino-shi, Tokyo, 180-8585 Japan
{shimizu.masafumi, nakazato.hikotoshi, seshake.hikaru}@lab.ntt.co.jp

*Abstract*— **We propose an architecture that optimizes system performance by adding resources cost-effectively and in a timely manner to reduce operation costs. This paper presents two methods for competing control, which is one of the requirements of service order processing (SO) systems. We compared the two methods based on ease of implementing device expansion, and a queuing model. It's hoped that this technology is appropriate for network operations systems (OpS) such as SO systems, rather than general applications to web/application servers.**

*Index Terms*— **service order processing, operation system, scale-out**

## I. Introduction

The next-generation network (NGN) [1] business service that unifies telephone, data communication, and streaming broadcasts has begun, and therefore, as the number of customers increases, the number of network devices that constitute the NGN, such as router servers, will also increase. Consequently, a large scale change of the network operations system (OpS) itself, which manages the network devices, is necessary. For better cost-efficiency of OpS equipment, early NGN systems were built with minimal configurations so that future expansions would proceed progressively and economically, and would comply with system load. However this increases the operating cost due to the expansion of OpS facilities based on the increase in the number of network devices. OpS facilities will occasionally need to be expanded in the future in order to obtain further cost-efficiency, and therefore, a method to provide quick and economical facility resources and to optimize system performance is being examined.

Networks are used to supply various services to customers. After receiving a customer's application for such services and securing the necessary network facilities, the configuration finally needs to be set into the network elements (NEs). The service order processing system (SO system) handles the processing of the series of steps from the receipt of the service application until the start of the service.

One of the requirements of the SO system involves the setting of identical NEs in response to applications from multiple customers and based on SO done at the same time. If a configuration abnormality (e.g., a configuration fault) occurs, a structure that does not carry out processing for multiple configurations at the same time for identical NEs is necessary in order to avoid roll-back processing. To implement such a structure, the system is equipped with a competing control function that can be carried out in a service order unit at the time the configuration is set into the NE. Additionally, the SO system redundancy architecture is composed of the pair of a single active server and a standby server as the minimum resource architecture.

When adopting this system architecture, following point is important when the system exceeds the performance requirements. The point is that the system may comply with traffic rules and proceed to start a service for a customer that shall not exceed the system performance when the load factor of the SO system increases in response to temporary increases in the number of service applications from customers. To prevent degradation of the system response due to the increased amount of processing because of the traffic rules, the system may take a long time to complete the processing of a large number of applications for information services. We refer to this as problem 1. We consider it necessary to establish the architecture as follows in order to solve this problem. First, we aim to keep the time to comply with traffic rules during temporary increases in load as short as possible by dynamically improving the performance.

This paper examines an architecture designed to optimize system performance by building more resources quickly and economically while aiming at improved scalability of the SO system.

## II. Related work

The number of SOs that process tasks continue to increase as services are added and customer numbers grow. Furthermore, demand is increasing for quick processing of the SO system. Work is proceeding on SO systems that meet these needs and that reach a target level of efficiency [2][3]. However, when the hardware performance of a system is limited, and when the performance requirements of a system are exceeded due to the increase in SO input, it is necessary to expand the facilities of the SO system itself. Even when

978-3-901882-50-0 

demand exceeds the performance requirements, it is desirable to build more servers quickly without stopping the servers in operation.

Scale-out architecture enables load balancing by distributing processing to multiple physical servers through a load balancer at high speed. Google's cloud service environment is said to be the most advanced platform as a technology that enables more servers to be added quickly. It uses an elemental technology called MapReduce [4]. However, this kind of technology is intended for application to web/application servers, rather than to OpS such as SO systems.

In regard to NE technology, work is proceeding on configuration replacement and rollback [5] [6] by NE vendors, although it is not sufficient for the requirements of the SO system.

## III. SCALE-OUT ARCHITECTURE

In this section, we present the details of the proposed scale-out architecture that makes it possible to optimize system performance and meet the SO system requirements.

The prerequisite of the scale-out architecture is that the single load-balancing unit sends SO data to the SO processing units. The multiple SO processing units are divided into two or more components. The architecture executes the SO processing depicted in Fig. 1.

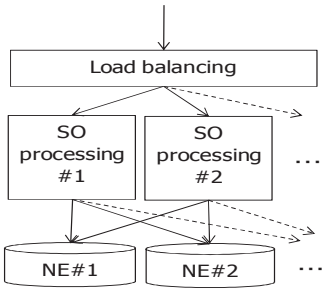


Fig. 1. Scale-out architecture

### A. Method of competing control

Here, we describe the method of competing control, which is one of the requirements of the SO system.

When the load of the system increases, the performance is optimized by expanding the SO processing unit. As mentioned in the preceding section, simultaneous execution of multiple SO processes in target NEs with similar settings is not possible in the SO system. Two methods of executing competing control are possible: (a) monitoring the configuration conditions of the SO processing unit, and (b) carrying out parallel computation by dividing the SO processing unit into two or more parts in every targeted NE configuration. We examine these two methods here.

In method (a), upon receiving multiple SO execution orders, each SO is divided into two or more SO processing units by means of a load algorithm. Then, the SO configuration is set into each NE after observing the configuration state of all SO processing units so as to not execute multiple configuration processes in identical NEs. In method (b), upon receiving multiple SO execution orders, each SO is distributed to the SO processing units that manage the specific NEs by means of a sorting algorithm. Then, the SO configuration is set into each NE.

These two methods are evaluated on the basis of three factors: load homogeneity, ease of execution of the competing control function, and ease of facility expansion. In method (a), the load homogeneity of each SO processing unit is guaranteed; however, functional implementation of this method is difficult when it is necessary to observe the configuration state of all SO processing units. Also, the monitoring load increases when the number of SO processing units increases. In method (b), implementing the competing control of the SO processing unit is easy; however, when the number of requirements for a specific target NE increases, the load of the identified SO processing unit also increases, and load homogeneity is not preserved. In addition, an implementation method that facilitates environmental changes of the SO system for the NE expansion is required. In method (a), this is required with the load balancing unit and the status monitoring unit. In contrast, in method (b), this is done only with the load balancing unit. Therefore, method (b) is preferable considering its ease of implementation.

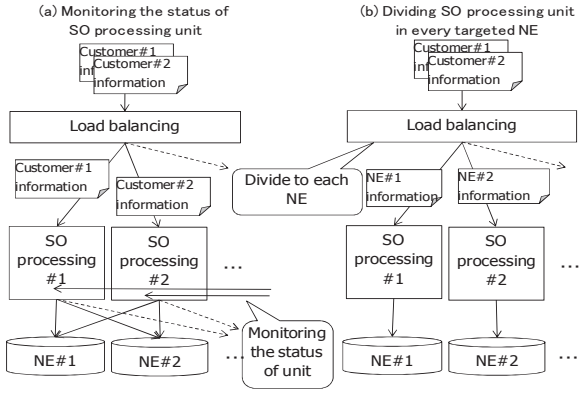


Fig. 2. Competing control method using scale-out architecture

### B. Queuing model

Next, we propose two queuing models, which we evaluated using queuing theory [7] [8] [9] based on the processing time (Fig. 3) and on traffic bottleneck. As a precondition of the queuing model, we use a queuing network consisting of multi-stage queues. There are three types of service, as mentioned in the previous section. Model (a) consists of the load balancing unit, SO processing unit, and synchronization processing unit, and model (b) consists of the load balancing unit and the SO processing unit. The SO processing unit conducts parallel processing in unit k; the processing time for each node in both model (a) and (b) is equal.

*1) Queuing model (a):*

The average waiting time of the synchronization processing unit is given by using polling models [10] [11] in which this execution unit is cycling through multiple queues. Thus, the average waiting time is given by E[W], and the average service time (synchronization processing unit) is given by $T_{a3}$. In this case, we use the limited service of polling models. Therefore,

the limited service unit skips the arriving queue if the arriving queue is empty, or it processes only one task and moves to the next queue if the arriving queue is not empty.

Then, the arrival rate of the load balancing unit is given by $\lambda$ under the Poisson distribution, and the average service time (load balancing unit) is given by $T_{a1}$. The arrival rate of SO processing unit #k is given by $\lambda_{ak}$, and the average service time (each SO processing unit) is assumed to be constant and is given by $T_{a2}$.

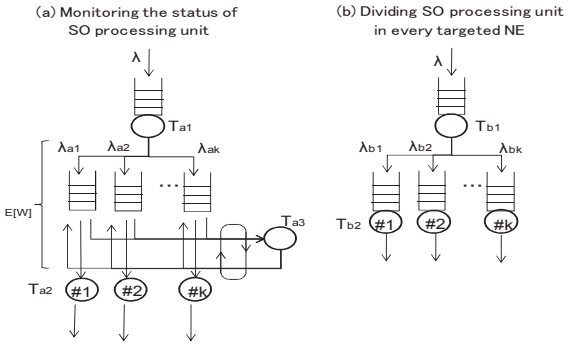


Fig. 3. Queuing model

N is the number of parallel SO processing units, $b^{(2)}$ is the second moment distribution function of service time for queuing k, $\delta^2$ is the variance distribution function of walking time for queuing k, and $r$ is the arithmetic mean distribution function of walking time for queuing k. The relationship between $\rho$, $\lambda_{bk}$ and $\lambda$ is as follows.

$$\lambda = \sum_{k=1}^{N} \lambda k \qquad (1) \qquad \rho_k = \frac{\rho}{N} = \lambda b \qquad (2)$$

Thus, the average processing time $T_{atotal}$ of queuing model (a) can be represented as follows.

$$T_{atotal} = \frac{\lambda T_{a1}}{1-\lambda T_{a1}} \cdot T_{a1} + T_{a1} + E[W] + T_{a2} \qquad (3)$$

$$E[W] = \frac{\delta^2}{2r} + \frac{N\lambda b^{(2)} + r(N+\rho) + N\lambda\delta^2}{2(1-\rho-N\lambda r)} + T_{a3} \qquad (4)$$

The performance of the SO processing unit is expected to increase with the number of parallel processes. Therefore, we first evaluated the variation when the number of parallel SO processes increased.

We used this variation evaluation to evaluate the following parameters.

TABLE I. PARAMETERS

| parameter | value | parameter | value |
|---|---|---|---|
| $\rho$ | 0.9 | $T_{a1}$ | 0.9 |
| N | 1~10 | $T_{a2}$ | 0.9 |
| r | 0.1 (case 1-a)<br>0.01 (case 2-a) | $T_{a3}$ | 0.9 (case 1-a)<br>0.09 (case 2-a) |
| $b^{(2)}$ | 1 | $\delta^2$ | 0.01 |

The results of the evaluation are shown in Fig. 4. The graph on the left plots the total processing time and the average waiting time (synchronization processing unit) when the $N$ value is changed from 1 to 10 for $T_{a1} = 0.9$, $T_{a2} = 0.9$, $T_{a3} = 0.9$, and r =0.1 We call this case 1-a.

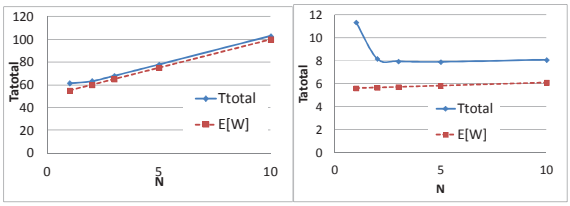


Fig. 4. Total processing time for number of parallel SO processing units (left: case 1-a, right case 2-a)

The evaluation results described in the previous section led to the concern with case 1-a that model (a) would become a bottleneck when the number of requirements for the synchronization processing unit increased. However, the left graph in Fig. 4 clearly shows that when the parallel processing increased, the speed of completion also increased, contrary to our expectation.

Next, we changed the parameters that give the synchronization processing unit high performance. We call this case 2-a.

The right graph in Fig. 4 shows the results of evaluating case 2-a. The total processing time and the average waiting time (synchronization processing unit) are plotted when the $N$ value was changed from 1 to 10 for $T_{a1} = 0.9$, $T_{a2} = 0.9$, $T_{a3} = 0.09$, and r = 0.01.

In case 2-a, the evaluation results indicate that the performance of model (a) improved when the number of parallel SO processing units was doubly increased. However, the average waiting time E[W] increased when the number of parallel SO processes increased, so $T_{atotal}$ did not decrease, contrary to our expectations.

These results indicate that if the synchronization processing unit has high performance, the average processing time becomes faster as the number of SO processing units increases in order to complete the processing. However, the synchronization processing unit can easily become a bottleneck.

*2) Queuing model (b):*

This is shown in the M/M/1 model of k+1. The arrival rate of the load balancing unit is given by $\lambda$ under the Poisson distribution, and the average service time (load balancing unit) is given by $T_{b1}$. The arrival rate of SO processing unit #k is given by $\lambda_{bk}$, and the average service time (of each SO processing unit) is assumed to be constant and is given by $T_{b2}$.

Thus, the average processing time $T_{btotal}$ of queuing model (b) can be represented as follows.

$$T_{btotal} = \frac{\lambda T_{b1}}{1-\lambda T_{b1}} \cdot T_{b1} + T_{b1} + \frac{\lambda_{bk} T_2}{1-\lambda_{bk} T_2} \cdot T_2 + T_2 \qquad (5)$$

The evaluation results in the previous section led to the concern that model (b) would become a bottleneck when the number of requirements for specific target NEs increased. Thus, we evaluated the variation in two cases where the arrival rate of a particular NE (SO processing unit # k) was large or small when the average processing times of $T_{b1}$ and $T_{b2}$ were changed. The following parameters were used in the evaluation.

TABLE II. PARAMETERS

| parameter | value | parameter | value |
|---|---|---|---|
| $\lambda$ | 0.96 | $T_{b1}$ | 0.09 (case 1-b)<br>0.09~0.9 (case 2-b) |
| $\lambda_{bk}$ | 0.02 or<br>0.9 | $T_{b2}$ | 0.09~0.9 (case 1-b)<br>0.09 (case 2-b) |

The results of this evaluation are shown in Fig. 5. The left graph plots the total processing time when the $T_{b2}$ value was changed from 0.09 to 0.9 for $T_{b1}$ = 0.09 (case 1-b). The right graph plots the total processing time when the $T_{b1}$ value was changed from 0.09 to 0.9 for $T_{b2}$ = 0.09 (case 2-b).

The left graph in Fig. 5 left shows that when an input load is concentrated at a particular NE, the processing is completed at a lower speed, which is based on the increase in SO processing unit time $T_{b2}$. Also, the right graph shows that regardless of the load uniformity for the target NEs, it is clear that it uses a lower speed to complete the processing, and the speed is based on the increase in load balancing unit time $T_{b1}$.

These results indicate that the load growth in the load balancing unit can easily become a bottleneck, and when the input load is concentrated at a particular NE, the SO processing unit becomes a bottleneck.

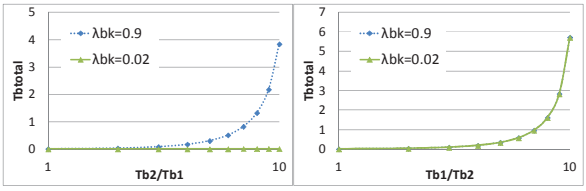


Fig. 5. Total processing time for processing time ratio of Tb1 and Tb2 (left: case 1-b Tb2 > Tb1, right: case 2-b Tb2 < Tb1)

## IV. CONSIDERATION

We conducted a comparative evaluation of the proposed system architecture from the viewpoint of scalability, which refers to the installation cost and the effectiveness with regard to problem 1 as an implementation model that applies Scale-out technology. We hold the definition of "scalability" to be the characteristic of a system which, when there is a deficiency in processing power, can be augmented in a timely fashion. To be more specific, this definition does not mean that a high-powered system is ready and waiting from day one; rather, it means that we can, in response to the load on the system, economically make upgrades to the system in stages to cope with that load. We assessed the scalability of the proposed system architecture with regard to problem 1. We found that for problem 1, when there is an increase in inputs to an NE that is being managed by a VM deployed on a given physical server, a live migration method is applied to increase efficiency. This enables us to achieve a rapid expansion of resources in the proposed system architecture.

## V. CONCLUSION

We proposed a scale-out architecture for reducing the provisioning and operation costs associated with large-scale SO systems. We first presented the details of two proposed scale-out architectures. We also evaluated these architectures based on three factors: ease of implementing device expansion, ease of executing a competing control function, and load homogeneity. Method (b) was found to be preferable due to its ease of implementation. We also proposed two queuing models by using queuing theory to evaluate both methods based on the processing time, and we clarified what units can become bottlenecks. Next, we evaluated the proposed system architecture based on scalability and clarified its effectiveness. In the future, we will carry out more advanced evaluations based on the implementation model.

## REFERENCES


[1] "NGN: The Key to a Ubiquitous Broadband Society," https://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr200712lc1.html

[2] K. Sakata, T. Kimura, and Y. Otsuka, "Reducing excess processes of router control requests in OSS for managing large-scale IP network," *Proc. of IEEE International Workshop on Management of Emerging Networks and Services (IEEE MENS 2010) in conjunction with IEEE Globecom 2010*, pp. 148-159, Dec. 2010.

[3] A. Ohizumi, T. Furukawa, M. Ogawa, and K. Yamane: "A Method of Processing Service Ordering Data for Various IP-Network Services," APNOMS2002, pp.148-159, Sept. 2002.

[4] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December 2004.

[5] "Configuration Replace and Configuration Rollback," http://www.cisco.com/en/US/docs/ios/fundamentals/configuration/guide/cf_config-rollback.pdf

[6] "JUNOS Technical Documentation," http://www.juniper.net/techpubs/en_US/junos11.4/topics/reference/command-summary/rollback.html

[7] L. Kleinrock, "Queueing Systems Vol. I: Theory," Wiley-Interscience, 1975.

[8] H. Kobayashi, "Modeling and Analysis 1," Addison-Wesley, 1978.

[9] R. Jain, "The Art of Computer Systems Performance Analysis," John Wiley & Sons, 1991.

[10] H. Takagi, "A Survey of Queueing Analysis of Polling Models," Data Communication Systems and Their Performance, L.F.M. de Moraes. E. de Souzae Silva & L.F.G. Soares, pp.277-295, North-Holland, 1988.

[11] T. Takine, Y. Takahashi and T. Hasegawa, "Analysis of an Asymmetric Polling System with Single Buffers," Performance '87, Brussels, pp.241-251, 1987.