Modeling and Evaluation of Serial Multicast Remote Procedure Calls (RPCs)

Sang-Hoon Kim, *Student Member, IEEE,* Jin-Soo Kim, *Member, IEEE,* and Seungryoul Maeng

***Abstract*—The need for serial multicast remote procedure calls (RPCs) arises in many distributed systems to avoid network bottlenecks and high-latency links, especially when the caller wants to send a large amount of data to multiple callees. In this paper, we present a mathematical model of serial multicast calls in order to understand their performance behavior. Through our evaluations on the real platform, we find our model is very effective in predicting the performance of serial multicast RPCs.**

***Index Terms*—Serial multicast, remote procedure calls (RPCs), model, distributed systems.**

I. Introduction

SINCE the concept of Remote Procedure Call (RPC) was introduced at least as far back as 1976, it has been widely used to reduce system complexity and development cost in building distributed systems. Especially, many distributed file systems, such as Network File System (NFS), Andrew File System (AFS), and Coda, have been built on top of the RPC layer.

In RPC, the caller passes a series of arguments to the callee and waits until the response is received from the callee. Besides this simple request / reply calling pattern, modern distributed file systems require more complicated *multicast* calls where the same RPC request is delivered to multiple callees simultaneously.

In particular, Google File System (GFS) has recently suggested a new possibility of novel multicast calls; instead of sending a RPC request to multiple callees at once, each callee forwards the RPC request to the "closest" callee in the network topology that has not received it. GFS has demonstrated that this *serial multicast* calls are effective in avoiding network bottlenecks and high-latency links [1]. In spite of these advantages, there is no known RPC layer which supports serial multicast calls, not to mention that their performance behavior is not fully investigated.

We have built a new RPC system called FlexRPC to ease the development of sophisticated modern distributed file systems [2]. As far as the authors are aware, FlexRPC is the first RPC system which natively provides serial multicast calls in the RPC layer. In this paper, we present a mathematical model of serial multicast calls and evaluate their performance on the real platform.

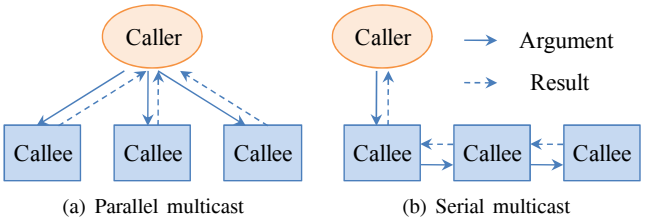
Manuscript received December 12, 2008. The associate editor coordinating the review of this letter and approving it for publication was C. Douligeris.

S.-H. Kim and S. Maeng are with the Department of Computer Science, KAIST, Daejeon 305-701, South Korea (e-mail: {sanghoon, maeng}@calab.kaist.ac.kr).

J.-S. Kim (corresponding author) is with the School of Information and Communication Engineering, Sungkyunkwan University, Suwon 440-746, South Korea (e-mail: jinsookim@skku.edu).

This work was supported by the Korea Science and Engineering Foundation(KOSEF) grant funded by the Korea government(MEST) (No.R01-2007-000-11832-0).

Digital Object Identifier 10.1109/LCOMM.2009.082103



(a) Parallel multicast (b) Serial multicast

Fig. 1. Multicast calls in RPC.

II. Multicast Calls

The need for multicast calls arises in many distributed algorithms and systems. For example, many replication-based distributed file systems use multicast calls heavily to distribute the whole or part of files across several servers to enhance data availability against component failures. P2P-based distributed file systems probe available nodes and files using multicast calls. The Paxos distributed consensus algorithm requires multiple nodes to be contacted to perform an operation.

We classify multicast calls into *parallel multicast* and *serial multicast* according to the path in which the request and the reply are delivered, as illustrated in Figure 1. We call the number of destination callees *the multicast degree*.

In parallel multicast, the RPC request / reply is sent to / received from each callee in parallel. Parallel multicast was introduced in MultiRPC [3] and has been used extensively in many distributed systems. Although it is simple to implement and runs at low latency for small arguments, it consumes outbound bandwidth quickly for bulk data transfer, resulting in severe congestion in network links. On the contrary, data are transferred in a pipelined fashion along a *delivery chain* of callees in serial multicast. Such a scheme lowers outbound traffic of the caller and distributes traffic pressure over the delivery chain.

III. Modeling Serial Multicast

We present a mathematical model of serial multicast implemented in our FlexRPC layer. As described in Section I, developing a proper model and understanding the performance characteristics of serial multicast is crucial to the development of distributed applications.

Let A and R be the size of argument and the size of result of an RPC call, respectively. Assume that a caller C invokes a serial multicast call to n callees (denoted as $S_1, S_2, \cdots, S_n$), i.e., the multicast degree is n. The processing time for the RPC

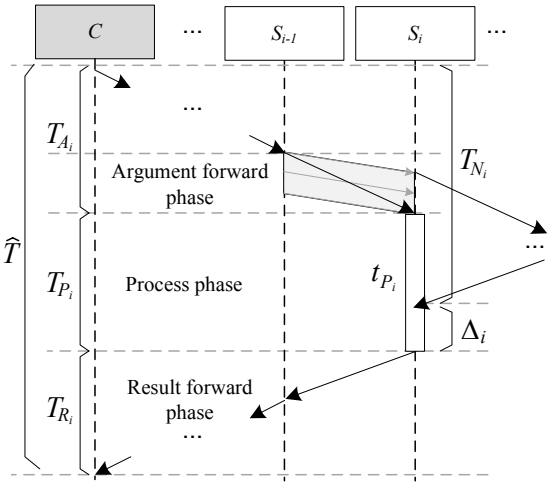


Fig. 2. Phases of serial multicast at the callee S_i.

request in the callee S_i is given by t_{p_i}. The caller and callees are connected via full-duplex network links whose bandwidth is B. We also assume the argument and the result are split into the fixed-size packets whose size is P and they are forwarded to the other node on a per-packet basis.

We introduce a *relay efficiency* α_i which represents the system efficiency to relay the received packet to the next node for the callee S_i. If S_i receives a packet in t_{recv_i} and forwards it to the next node in t_{fwd_i}, the relay efficiency is defined as $\alpha_i = t_{recv_i}/t_{fwd_i}$. For an ideal pipelining, $\alpha_i = 1$ for all the nodes involved in serial multicast. In reality, however, α_i is less than 1 due to the contention in shared resources to receive and send packets concurrently. The value of α_i depends on the implementation and system efficiency, and tends to become closer to 1 as the callee is positioned farther from the caller in the delivery chain. This is because as packets are transmitted along the delivery chain, the amount of inbound traffic arrived at each callee continuously diminishes due to the imperfect relay efficiency in previous callees, thus reducing the contention in the system.

As Figure 2 depicts, we divide the whole process of serial multicast into three phases. A callee S_i enters the *argument forward phase* when S_{i-1} (the caller C for S_1) begins to forward the argument to S_i. After receiving the entire argument, S_i moves to the *process phase*, and handles the incoming RPC request. If the processing is finished, S_i advances to the *result forward phase*, where the result of S_i and the results from $S_{i+1}, \cdots, S_n$ are sent back to S_{i-1}. The result forward phase ends when all the results from S_n to S_1 are returned to the caller C.

In an ideal condition, the argument and the result are divided into packets and each packet is transmitted in $t_0 = P/B$. Let t_A and t_R be the total time to transmit the entire argument and the result, respectively. Then, t_A and t_R are given by

$$t_A = \left\lceil \frac{A}{P} \right\rceil \cdot t_0 \quad (1)$$

$$t_R = \left\lceil \frac{R}{P} \right\rceil \cdot t_0 \quad (2)$$

A callee S_i starts the argument forward phase when S_{i-1} receives the first packet from S_{i-2} and leaves the phase after t_A/α_{i-1}. As noted above, t_A is divided by α_{i-1} to model the prolonged argument forward phase at S_{i-1} due to the system overhead. By the following recurrence relation, we can obtain T_{A_i} which represents the total elapsed time to receive the entire argument at S_i.

$$\begin{aligned} T_{A_1} &= t_A \\ T_{A_i} &= \frac{T_{A_{i-1}}}{\alpha_{i-1}} + t_0 \\ &= t_A \prod_{j=1}^{i-1} \frac{1}{\alpha_j} + t_0(i-1) \quad (\text{for } i > 1) \end{aligned} \quad (3)$$

By definition, T_{P_i}, the time needed to process the RPC request at S_i coincides with t_{p_i}. Therefore,

$$T_{P_i} = t_{p_i} \quad (4)$$

The results are buffered, then forwarded to the subsequent node rather than being transferred in a pipelined fashion. The data to transfer from S_i in the result forward phase are $(n-i)$ results from $S_{i+1}, \cdots, S_n$ plus its own one. Therefore, T_{R_i}, the time needed for the result forward phase in S_i can be described by the following recurrence relation.

$$\begin{aligned} T_{R_1} &= n \cdot t_R \\ T_{R_i} &= (n-i+1)t_R + T_{R_{i-1}} \\ &= \sum_{j=1}^{i} (n-i+j)t_R \\ &= t_R \left[\frac{i(2n-i+1)}{2} \right] \quad (\text{for } i > 1) \end{aligned} \quad (5)$$

Equation (5) assumes that S_i finishes its process phase and sits idle at the time the result is delivered from S_{i+1}. However, this is not always the case as can be seen in Figure 2. Let T_{N_i} be the time S_i spends until the result is returned from S_{i+1}. In Figure 2, the result from S_{i+1} arrives at S_i while the process phase is still in progress, i.e., $\Delta_i = T_{A_i} + T_{P_i} - T_{N_i} > 0$. As a consequence, we can see that T_{R_i} is delayed by Δ_i.

Formally, the total amount of delay, Δ, caused by variations in $T_{A_i} + T_{P_i}$ in each callee can be estimated as follows:

$$\begin{aligned} \Delta_n &= 0 \\ \Delta_i &= \delta_i \cdot H(\delta_i) \quad (\text{for } i \geq 1) \end{aligned} \quad (6)$$

$$\Delta = \sum_{i=1}^{n} \Delta_i \quad (7)$$

where $H(x)$ is the unit step function and

$$\delta_i = T_{A_i} + T_{P_i} - T_{N_i} \quad (8)$$

$$\begin{aligned} T_{N_n} &= T_{A_n} + T_{P_n} \\ T_{N_i} &= T_{N_{i+1}} + \Delta_{i+1} + (n-i) \cdot t_R \\ &= T_{N_n} + \sum_{j=i+1}^{n} \Delta_j + \sum_{j=1}^{n-i} j \cdot t_R \quad (\text{for } i \geq 0) \end{aligned} \quad (9)$$

In Equation (9), we have intentionally defined the value of T_{N_0}, because the caller can be regarded as the 0-th callee. In fact, T_{N_0} is equal to $\widehat{T}$ which represents the minimum

required time to process the entire serial multicast request. From Equation (3), (4), (5), (7), and (9), $\widehat{T}$ is expressed as

$$\begin{aligned}\widehat{T} &= T_{N_0} = T_{N_n} + \sum_{i=1}^{n} \Delta_i + \sum_{i=1}^{n} i \cdot t_R \\ &= T_{A_n} + T_{P_n} + T_{R_n} + \Delta \\ &= t_A \prod_{i=1}^{n-1} \frac{1}{\alpha_i} + t_0(n-1) + t_{p_n} + t_R \left\lceil \frac{n(n+1)}{2} \right\rceil \\ &\quad + \Delta \end{aligned} \tag{10}$$

The time to complete a serial multicast call can be minimized by reducing each term in Equation (10). In particular, if t_{p_i}'s are identical in all callees which is very common in practice, Δ can be ignored. In addition, when the relay efficiency is ideal, i.e., $\alpha_i = 1$ for all S_i, $\widehat{T}_{ideal}$ is given by

$$\widehat{T}_{ideal} = t_A + t_0(n-1) + t_{p_n} + t_R \left\lceil \frac{n(n+1)}{2} \right\rceil \tag{11}$$

Note that $\widehat{T}_{ideal}$ serves as the lower bound of the latency in any serial multicast implementation.

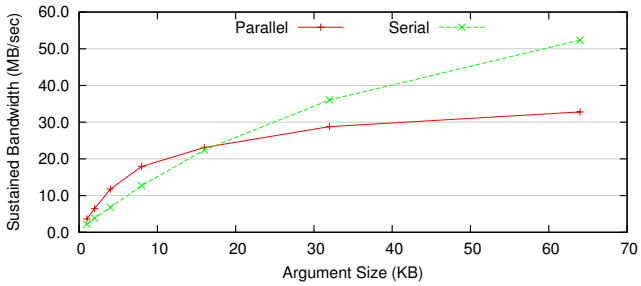
IV. Evaluation

The evaluation has been performed on eight Linux nodes running the Linux kernel 2.6.18-1, connected by a Netgear GS748TS Gigabit Ethernet switch. A node is dedicated to the caller and the rest are used as the callees. We measured the latency of serial multicast calls varying the argument size, the result size, and the multicast degree. Figure 3 summarizes our evaluation results.

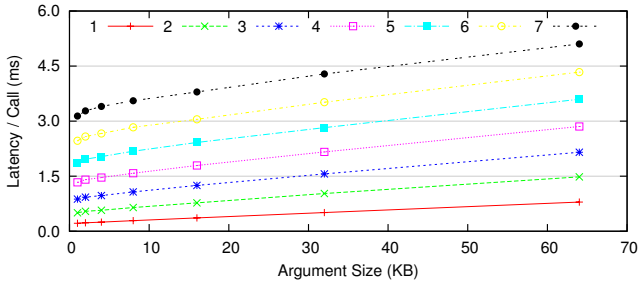
First, we compare the sustained bandwidth of parallel and serial multicast RPCs, varying the argument size from 0 KB to 64 KB. As Figure 3(a) depicts, we can observe that the serial multicast outperforms the parallel multicast in handling bulk argument whose size is larger than 16 KB. This is because as the argument size increases, the outgoing link of the caller becomes saturated during the parallel multicast calls.

Figure 3(b) illustrates that the argument size (A) linearly affects the latency, as expected from Equation (10). The slope is largely determined by the network bandwidth ($1/B$), and it is slightly increased due to the worsen relay efficiency as the multicast degree grows. The result size also affects the latency linearly under the given multicast degree, as Figure 3(c) shows. As the multicast degree increases, however, the slope gets steeper in proportion to $O(n^2)$ as suggested by Equation (10).

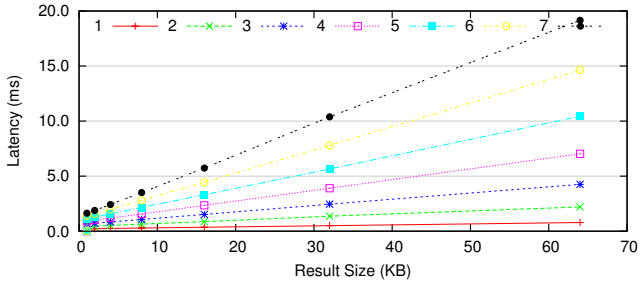
On our evaluation platform, we found that $t_0 = 154.37(\mu s)$, $t_{p_n} = 140.40(\mu s)$, $t_A = 0.0089 \cdot A(\mu s)$ and $t_R = 0.0089 \cdot R \cdot n(n+1)/2(\mu s)$. Empirically, we also obtained the following relay efficiencies: $\alpha_1 = 0.60$, $\alpha_2 = 0.77$, $\alpha_3 = 0.85$, $\alpha_4 = 0.88$, $\alpha_5 = 0.93$, and $\alpha_6 = 0.96$. With these parameters, we predict the latency of serial multicast calls in Figure 3(d) (labeled as MODEL), along with the measured latency (labeled as ACTUAL) and the lower bound given by Equation (11) (labeled as IDEAL) when both the argument size and the result size are 32 KB. We can verify that our model presented in this paper is very effective in predicting the performance of serial multicast RPCs.



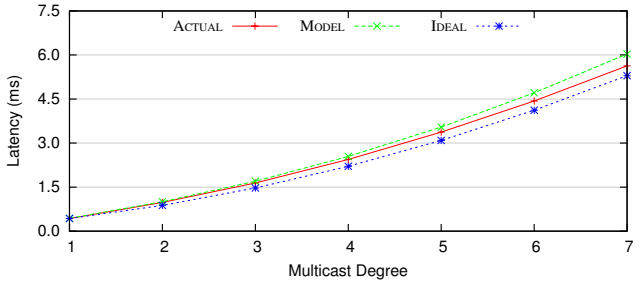
(a) Sustained bandwidth: parallel multicast vs. serial multicast (multicast degree = 3, result size = 0 KB)



(b) The latency of serial multicast calls varying the argument size and the multicast degree (result size = 8 KB)



(c) The latency of serial multicast calls varying the result size and the multicast degree (argument size = 8 KB)



(d) Comparison of the ideal latency, the latency expected by the model, and the actual latency (argument size = 32 KB, result size = 32 KB)

Fig. 3. The measured performance of serial multicast RPCs.

References

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *Proc. 19th ACM Symp. on Operating Systems Principles (SOSP'03)*, 2003.

[2] S.-H. Kim, Y. Lee, and J.-S. Kim, "Flexrpc: a flexible remote procedure call facility for modern cluster file systems," in *Proc. 9th IEEE International Conference on Cluster Computing (CLUSTER'07)*, 2007, pp. 257–284.

[3] M. Satyanarayanan and E. H. Siegel, "Parallel communication in a Large distributed environment," *IEEE Trans. Computers*, vol. 39, no. 3, Mar. 1990.