J. of Basic Sciences, Cheju Nat. Univ.
10(1), 7~13, 1997

기초과학연구 제주대학교
10(1), 7~13, 1997

TDMA 구현 이더네트에 기반한 실시간 RPC 시스템의 설계

A design of Real-Time RPC based on TDMA-implemented Ethernet

이 정 훈*, 이 봉 규*, 김 성 백**
*제주대학교 자연대학 전산통계학과 **제주대학교 사범대학 컴퓨터교육과
Tel: +32-64-54-3594, e-mail:jhlee@venus1.cheju.ac.kr

초 록

본 논문은 요청-확인-응답 시맨틱을 갖는 실시간 원격 프로시쥬어 호출을 효율적으로 지원할 수 있는 통신 시스템을 설계하고 그 성능을 평가한다. 이 통신 시스템은 시분할 다중 접근 방식이 구현된 이더네트를 기반으로 하고 있으며 원격 프로시쥬어 호출에 관련된 메시지들을 그들의 특성에 따라 스케쥴하여 서버의 스케쥴과 쉽게 동기화시킨다. 서버와 통신 시스템 스케쥴 동기화에 의해 서버의 수행시간을 효율적으로 사용할 수 있으므로 실시간 원격 프로시쥬어 호출의 종료시한 만족도를 향상시킬 수 있다. 요청의 도착시간, 서비스 시간 및 여유 시간 등 다양한 환경 인자 설정에 의한 모의 실험에 의해 제안된 통신 시스템이 기존의 토큰 버스나 시분할 다중 접근 방식보다 실시간 원격 프로시쥬어 호출의 실시간 성능을 향상시킴을 보인다.

Abstract

This paper proposes and analyzes a communication mechanism capable of supporting real-time RPC(Remote Procedure Call) which is based on request-acknowledgment-reply semantic. As an enhanced protocol of TDMA-implemented Ethernet, the proposed CS(Communication System) dynamically schedules RPC-related messages according to their attributes, making a network schedule be easily combined with a server schedule. As RPC schedule produced by combining server schedule and communication schedule can obviate the waste computation time at the server, more real-time RPC requests can meet their timing constraints. Simulation result shows that the RPC over the proposed CS outperforms those over token bus and TDMA on the various network parameters, such as RPC arrival time, service time and slack.

1. Introduction

RPC is the traditional and wide-spread type of interprocess communication and most of the distributed applications and distributed system services are built on top of this facility. In the future, ODP's such as CORBA(Common Request Broker Architecture)[1] and TINA (Telecommunication Networking Architecture)[2] will be required to support diverse distributed real-time applications, hence, it is necessary to provide enhanced RPC mechanism capable of efficiently supporting real-time characteristics. Like other real-time services, the correctness of a real-time RPC depends on both the correctness of the computation result and the time the result is produced. In other words, the result is meaningful only when the computed result is transmitted back to the caller within deadline of the RPC. Therefore, when a server receives a request, it must check if it can complete execution and if the communication system can guarantee delivery of the result within the deadline of the request. If a server executes the request without the guarantee of communication system[3], the result may be transmitted back to the client after deadline of the RPC and the computation time of the server is wasted. In addition, server should be able to inform the client early that it cannot execute the request so that the client can send the rejected request to another server. For this purpose, the communication system should be able to reserve necessary amount of network time within the deadline of the request and to report to the server on the reserved time so that the server completes prior to the reserved network time. The reserved network time is the actual deadline for the server computation, and server can reschedule the requests afterwards, according to this deadline.

This paper is organized as follows: Section 2 describes a CS which can efficiently support RPC-style real-time communication satisfying above requirements. Section 3 designs a real-time RPC computation model based on the proposed CS. Section 3 evaluates the performance and finally Section 4 concludes this paper along with the description of future work.

2. Functionality of Communication System

An RPC generated by a client invokes server execution and transmission of three messages, namely, request(REQ), acknowledgment(ACK), and reply(REP). ACK is adopted for earlier acknowledgment or denial of the request to the client. The main function of CS is to schedule efficiently each type of message and provide server scheduler with the information on its schedule. As the RPC request arrives dynamically, the RPC request arrival time is not known in advance. This implies that CS scheduler should follow the dynamic scheduling policy such as EDF(Earliest Deadline First), aiming at guaranteeing to meet the deadlines of requests as many as possible. However, the server node generates ACK and REP only after REQ arrives. It makes each server node schedule at each arrival time of REQ by reserving network time for ACK and REP.

The proposed CS is based on TDMA protocol implemented on Ethernet type bus, because its guarantee mechanism is straightforward[4]. In addition, a *hand-over* mechanism can be easily added, which enables a node to get a part of slot time of another node. As generally known, in TDMA, time axis is divided into equally sized frames, and each of the frames is divided into equally sized slots assigned to individual nodes. The disadvantage of TDMA is that the large amount of time is wasted when nodes do not send messages. However, on TDMA implemented by software[5,6], it is possible to dynamically mediate the slot time (by hand-over function) on each arrival of RPC request. For this purpose, the proposed CS invites a new type of message, called hand-over request(HOR). Like REQ, HOR implicitly passes the access right to the network over to receiver node until the node receives the corresponding ACK. That is, part of a slot time owned by a node can be used by the receiver of REQ and HOR. ACK plays the following two roles: It informs the client whether server can execute the request within deadline(T-ACK) or not(N-ACK). It also informs the requester whether an HOR request has been accepted(H-ACK) by the receiver of hand-over request or not(HN-ACK).

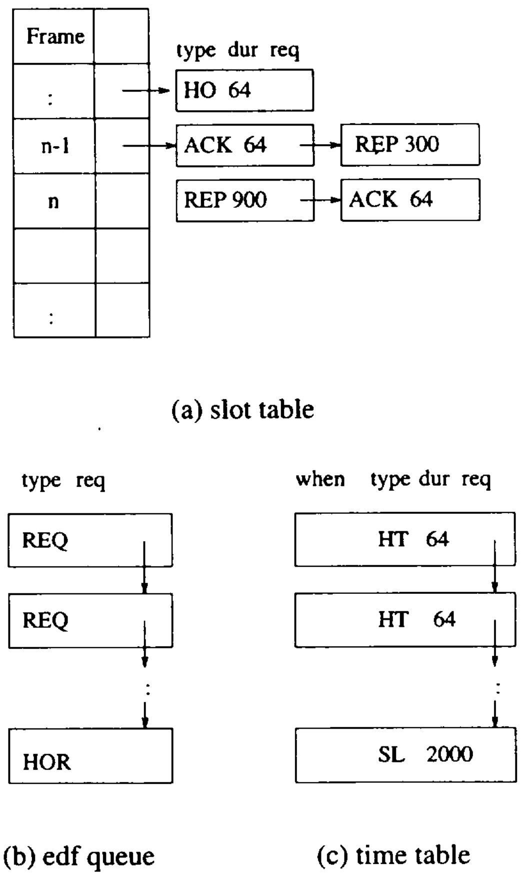

Figure 1 Data structure for each node

For the purpose of efficiently scheduling RPC requests, each node maintains a table for its own slot which contains reserved messages such as ACK and REP along with *edf queue* for REQ and HOR, as shown in Figure 1(a) and (b), respectively. Slot table contains the messages which have been reserved in advance and will be transmitted via its slot. The messages in edf queue can be sent only if the slot table entry of the frame is empty. In Figure 1, HO means part of slot time is handed over to another node. Each entry of both slot table and edf queue has *req* field which describes the characteristic of a request, such as deadline, required bandwidth, and so on. In addition, a node has time table by which its clock indicates to CS that it is time to send a message(See Figure 1(c)). In the figure, HT means that the time quantum is handed over from another node, while SL means its original slot. The last entry of time table is always SL and whenever the last entry is deleted, a new SL entry is inserted.

CS has three main functions as shown in Figure 2. *cs_send* is invoked on each of slot time and handed over time(Refer Figure 1(c)). Its function is to send the messages in its slot table of corresponding turn(when slot table entry is ACK or REP) or keep silent for the time duration handed over to another node(when slot table entry is HO), and then transmit messages in edf queue if remaining time is sufficiently available. In contrast, CS continuously calls *cs_receive* waiting

```
procedure cs_send()
  if (my slot time) {
     forall entries in the slot table
        if (type == Handed Over) keep silent;
        else send a message;
     for the remaining slot time
        send messages in the edf queue;
  }
  else /* my time handed over from another node */
     send corresponding message;
end procedure

procedure cs_receive() {
   receive message();
   switch message type
     case ACK, REP : send to the client;
     case HOR :
        if (requested amount is available)
           update slot table;
           send H-ACK;
        else send HN-ACK;
     case REQ:
        cs_reserve(ACK);  /* with hand-over */
        if (failure) send N-ACK ;
           return;
        send T-ACK;
        invoke the server scheduler;
end procedure

procedure cs_reserve() {
  if (the network time is available in its own slot)
     update slot table;
     return success;
  if (without hand-over) return failure
  within (predefined time)
     find unrequested node and slot in the range;
     send HOR to that node;
     wait for the reply;
     if (H-ACK is received) return success;
  return failure;
end procedure
```

Figure 2 Description of CS

for a new message. *cs_reserve* is invoked by server node when REQ arrives to reserve network time for ACK, as well as by the server scheduler when it attempts to reserve network time for REP. The caller of this procedure should specify *range* and *duration*. Then *cs_reserve* reserves as much time as *duration* within time interval specified by *range*. The procedure first checks if the sufficient slot time of its own in the specified range is available. If not, it sends HOR to the node within the range, one by one, until H-ACK is returned or predetermined time is elapsed. The requested node checks its slot table and sends back H-ACK(after updating its slot table) or HN-ACK. *cs_reserve* also provides reservation without hand-over so that the network time for REP can be reserved only out of the slot time of the server. Hand-over procedure for REP can block the serve scheduler, as CS sends HOR via edf queue.

With the functionalities described above, CS provides the following primitives:

- sendedf : inserts a message into edf queue.
- sendslot : sends a message via reserved slot.
- readfromcs : reads a message from CS.
- reserve : reserves a part of slot time for message. /* with or without hand-over */

Again with the support of CS, a real-time RPC can be designed as shown in Figure 3. A server executes an RPC request only when it can reserve network time for REP. After REP is reserved, the reserved network time becomes the actual deadline of the request. Afterwards, when another message arrives, the server will reschedule requests according to the actual deadline. A new request can be inserted to server queue as long as the execution of the request does not result in missing of actual deadlines of all requests pending in the server queue.

```
procedure rpc_client
    sendedf(REQ, reqdscr);
    wait for the acknowledgment;
    if (N-ACK received or timeout)   some action
    else wait for REP;
end procedure

procedure rpc_server
    waiting for REQ from any client;
    schedule the request;    /* edf scheduling */
    compute earliest finish time;
    reserve(REP);        /* without hand-over */
    if (success)
        slotsend(ACK);
        insert the reqdscr to server queue;
    else slotsend(NACK);
end procedure
```

Figure 3 RPC model based on the proposed CS

3. Simulation results

The performance of the proposed scheme has been measured via simulation using SMPL[7]. The simulation compares RPC over the proposed CS with RPC over token bus and normal TDMA. We have measured success ratio of the real-time RPC request according to request arrival time(Figure 4), service time(Figure 5) and finally slack(Figure 6). In the simulation, we assumed that the length of ACK and HOR is 64 bytes, that of REP is fixed to 500 bytes. The

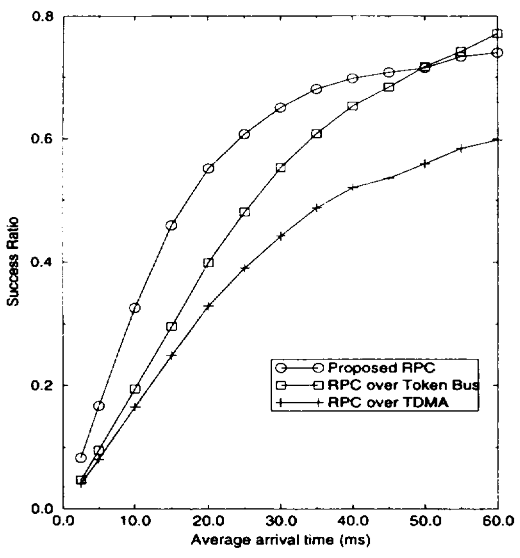

Figure 4 Success ratio according to arrival time

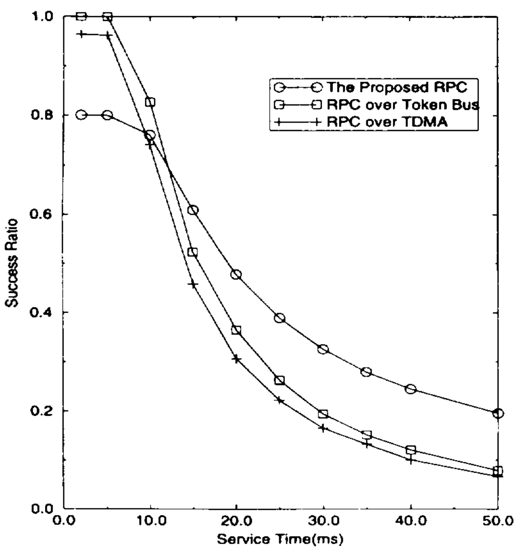

Figure 5 Success ratio according to service time

number of node is 4, requests arrive exponentially. The slot time for TDMA and maximum token holding time for token bus is 2 *ms* as in MARS system[5]. Token bus shows better performance when network load is low due to its flexible network access pattern and when service time is very small due to less damage resulted from waste of server execution time.

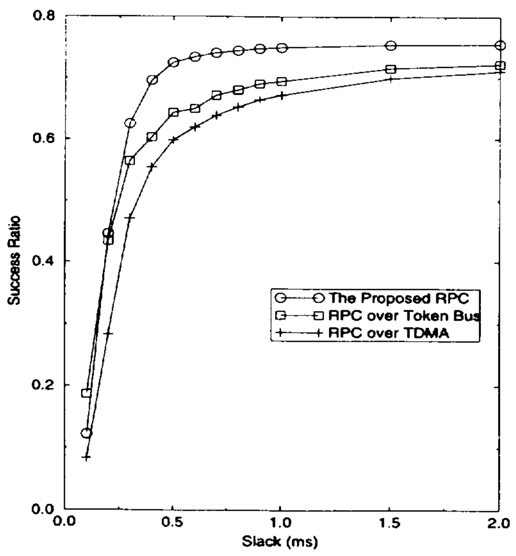

Figure 6 Success ratio according to slack

4. Conclusion and Future Work

In this paper, we have proposed CS capable of supporting real-time RPC. It's main function is to schedule RPC requests and provide synchronization with server scheduler. In this scheme, guaranteed RPC requests never fail to meet their deadlines. The obviation of server execution time mainly enhances the success ratio of the real-time RPC requests.

In addition, the following problems will be researched:

- Apply the functions designed for TDMA to another protocols such as token ring, CSMA/CD, and so on.
- Schedule efficiently when an request accompanies another RPC requestsuch as query to name server.
- Design an internetworking device such as bridge to support network-wide RPC.

참고문헌

[1] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, OM Document, No.91.12.1, December 1991.

[2] F. Dupuy and P. Graubmann et al., *DPE Phase 0.1 Specification*, TINA Consortium, December 1993.

[3] Marina Dao and Kwei-Jay Lin, "Remote procedure call protocols for real-time systems," *Proceedings of IEEE Euromicro Workshop*, pp. 216-223, 1991.

[4] Nicholas Malcomn, Wei Zhao, and Chris Barter, "Guarantee protocols for communication in distributed hard real-time systems," *Proceedings of IEEE INFOCOM: The Conference on Computer Communications*, pp. 1078-1086, 1990.

[5] Hermann Kopetz and et al., "Distributed fault-tolerant real-time systems of MARS," *IEEE Micro*, pp. 25-40, Feburary 1989.

[6] Junghoon Lee and Heonshik Shin, "A variable bandwidth' allocation scheme for Ethernet-based real-time communication," *Proc. of the First International Workshop on Real-Time Computing Systems and Applications*, pp. 28-32, December 1994.

[7] M. H. MacDougall, *Simulating Computer Systems: Techniques and Tools*. MIT Press, 1987.