A Dynamic Scheduling Service Model for Real-Time CORBA*

Luo Zhigang Tan Hao Liu Jinde
(College of Computer Science and Engineering, UEST of China Chengdu 610054)

Abstract The architecture of scheduling service specified in real-time CORBA is introduced and its limitation is analyzed in details. Through cooperation of client scheduler and server scheduler, the admission test that whether a new client is permitted to execute on a particular processing node is accomplished. Based on the admission test, a dynamic scheduling service model is proposed, which overcomes the limitation of current real-time scheduling service and can be applied to a dynamic open real-time CORBA system. At the end, a dynamic CORBA system based the scheduling service is presented.

Key words real-time CORBA; scheduling service; dynamic scheduling; admission test

1 Introduction

Traditional real-time systems are often highly optimized, but closed ones. They are really proprietary in that they are constructed for particular problem domains. Real-time software systems are extremely expensive and time-consuming to develop, validate, optimize, deploy, maintain and upgrade. Moreover, The use of specialized technologies of one kind or another makes the solutions are harder to adapt to new requirements, new technologies and new market opportunities. Real-time CORBA brings CORBA and real-time systems world together[1,2]. It provides real-time systems with lots of advantages supported by CORBA, such as portability, interoperability, maintainability, reusability, etc. Real-time CORBA will be beneficial to system builders and system integrators in markets such as telecommunications and manufacturing, where openness is gradually becoming crucial to competitiveness.

In order to simplify the development of real-time CORBA application, real-time CORBA specifies a scheduling service[2] that uses the real-time CORBA primitives to facilitate enforcing various fixed-priority real-time scheduling policies across the real-time CORBA system. The scheduling service abstracts away from the application some of the complication of using low-level real-time CORBA constructs, such as the POA policies. Application that uses an implementation of the scheduling service is assured of having a uniform real-time scheduling policy, such as global rate monotonic scheduling with priority ceiling, enforced in the entire system. The scheduling service uses "names" (strings) to provide abstraction of scheduling parameters (such as CORBA priorities). The application code uses these names to specify CORBA activities and CORBA objects. The scheduling service internally associates these names with actual scheduling parameters and policies. This abstraction improves portability with regard to real-time features, eases use of the real-time features, and reduces the chance for errors.

However, the scheduling service is designed to work in a "closed" CORBA system where the fixed priority policy is required for a static set of clients and servers. In such system, if some factors change, for instance a new client need to be added in the system, the whole system must be redesigned.

In order to adapt to dynamic environment and make real-time CORBA systems keep open, it is necessary to introduce dynamic scheduling service into real-time CORBA environment. Dynamic scheduling here doesn't mean that the scheduling algorithm must be dynamic scheduling one, such asEDF and MLF, but that scheduling service can adapt to dynamic environment. For instance, when a new client enters the system, the scheduling service can determine whether it is permitted to execute on a particular processing node with simple admission test. Based on the definition, a dynamic scheduling service is presented in the paper, which is implemented by cooperation between client scheduler and server

Received on Aug. 27, 2001
*The project is supported by National Defense Research Foundation

scheduler.

The remainder of the paper is organized as follows: Section 2 presents the related search work on the dynamic scheduling for real-time CORBA; Section 3 describes the dynamic scheduling scheme in detail; Section 4 gives the conclusion.

2 Related Work

Several researches have done in the area of dynamic scheduling for real-time CORBA systems. Among these researches, the representatives are: 1) TAO[3] ; 2) NRaD/URI RT-CORBAand 3) EPIQ[4,5].

The ACE ORB(TAO) developed at University of Washington in St.Lousis by Schmidt et. al[3] addresses many of the issues involved in providing support for hard and soft real-time applications in both static and dynamic environment. A strategized scheduling service framework for TAO have been developed to support dynamic scheduling[6]. Currently, TAO's scheduling service can perform both off-line and on-line feasibility analysis, in term of the scheduling algorithm which the developer choose. TAO relies on cooperation of off-line and on-line feasibility analysis to guarantee that the deadline of real-time tasks will be met on a particular object, and hence, guarantee the QoS for that object. But scheduling in TAO focuses on single CPU rather than the distributed scheduling problem.

Wolfe, et al, have developed a real-time CORBA at US Navy Research and Development Laboratories (NRaD) and University of Rhode Island (URI) [4]. NRaD/URI RT-CORBA presents a new CORBA global priority service[7]. It uses EDF within important level. The scheduler supports on-line scheduling and allows clients to be admitted at run-time. Due to the dynamic nature of NRaD/URI RT-CORBA, their on-line EDF scheduler doesn't offer the guarantee of an off-line RM scheduler and, unfortunately, can behave non-deterministically under heavy loads, thus providing only best-effort guarantees.

The EPIQ project defines an open real-time CORBA scheme that provides QoS guarantees and run-time scheduling flexibility[5]. EPIQ explicitly extends TAO's off-line scheduling model to provide on-line scheduling. In addition, EPIQ allows clients to be added and removed dynamically via admission test at run-time. The scheme is based on an open scheduling architecture[8], which provides a tow-level scheduling approach where each application is assigned a constant utilization server or total bandwidth server at upper level. At lower level, the OS scheduler maintains and schedules each of the servers by an EDF policy. The two-level scheduling approach allows developers of each real-time application to validate the schedulability of the application independently of other applications. The scheme relies on the two-level scheduling architecture, which is implemented by modifying OS kernel, so it is impossible to implement the scheme on conventional OS.

The above scheduling schemes focus on scheduling server object. However, the scheduling for a real-time CORBA application should include scheduling of client object and server object. So it is necessary for the schedulers of both sides, client and server, to cooperate to accomplish the scheduling of the whole real-time application. The dynamic scheduling scheme in the paper is just based on cooperation of client scheduler and server scheduler.

3 Design of A Dynamic Scheduling Service Model

In order to make Real-time CORBA scheduling service work in a dynamic open real-time CORBA system, the scheduling service should allow clients to be added and removed dynamically. The way of extension presented here is as follows: 1) introduces dynamic scheduling mechanism, which could refer to TAO's scheduling framework and EPIQ's admission test method for dynamic clients[9~11]; 2) makes client scheduler and server scheduler cooperate to negotiate the end-to-end timing constraints and reserve corresponding resource.

3.1 Dynamic Client

For a real-time CORBA system, adding a new client, it must guarantee the existed applications' timing constraints firstly. If the new client's timing constraints can be satisfied, it will be permitted to execute. Extending scheduling service from static to dynamic has a basic rule: application might use fixed

priority scheduling for part of the workload while the dynamic scheduling for only a subset of the RTOS_Priority range[9]. Conventionally, applications whose timing properties can be determined before run-time use off-line schedulability analysis to decide their schedule. Only those applications whose timing properties can not be determined before run-time use dynamic scheduling. This is because the cost of dynamic scheduling is expensive, especially for complex real-time system. To allow a new client request to be handled on-line, the model proposed in EPIQ is used here and shown as Fig.1.

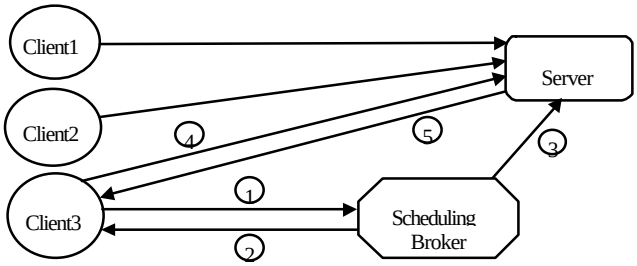

Fig.1 Dynamic Client

As an example, Fig.1 shows a scenario with three clients, a server and the server's scheduling broker. Client 1 and Client 2 can be validated and created corresponding schedule by off-line schedulability analysis before run-time. At run-time, the schedule is then used by the server's run-time scheduler to schedule requests from Client 1 and Client 2. Compared with Client 1 and Client 2, before server's run-time scheduler schedules request from Client 3, the request must first go through scheduling broker for schedulability analysis (i.e. admission test). The request from Client 3 can be processed as following steps (1~5 in Fig.1):

1) Client 3 requests scheduling broker to make admission test;
2) Once receiving admission test request from Client 3, the scheduling broker make schedulability analysis and then sends to clients the result whether being accepted or being refused;
3) If scheduling broker agrees to admit Client 3, it will notify server object to reserve resource;
4) If client has been admitted, it will begin to invoke remote operation;
5) The server returns the result.

3.2 The Cooperation of Client Scheduler and Server Scheduler

Fig.1 shows the run-time view of scheduling broker, but the details of admission test is ignored. Compared with TAO and EPIQ, admission test proposed in the paper is accomplished by cooperation of client scheduler and server scheduler. Fig.2 shows the details of cooperation among clients, client scheduler, server scheduler and server.

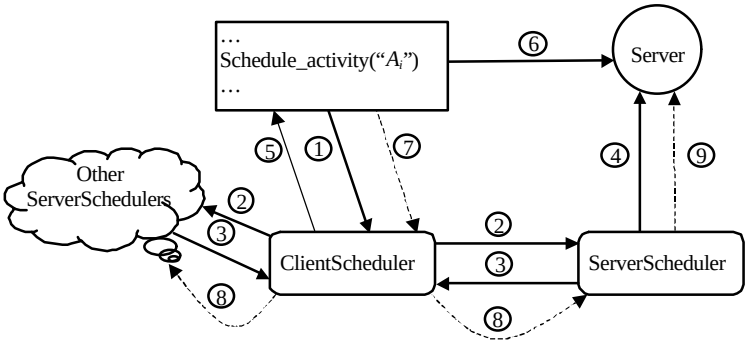

Fig. 2 Details of Admission Test

The process of scheduling a dynamic client's activity A_i can be described as follows:

1) When activity A_i arrives, client scheduler will be called for admission test (1 in Fig.2);

2) Client scheduler sends admission test request to each server scheduler (2 in Fig.2);

3) Each server scheduler makes admission test. If the system can meet on the timing constraints, the server scheduler will indicate client scheduler that the activity is permitted to execute;

4) Otherwise, the activity will be refused. (3 in Fig. 2);

5) If the activity is permitted to execute, the server scheduler will configure the server and reserve corresponding resources (4 in Fig.2);

6) Client scheduler collects all return results. If all admission test are successful, the activity A_i will be schedule (5 in Fig.2), otherwise the activity A_i is refused, meanwhile do as step 8;

7) The activity A_i begins to invoke the remote operation(6 in Fig.2);

8) While activity A_i terminates, it requests client scheduler to release resource (7 in Fig.2);

9) Client Scheduler requests each server scheduler to release resource (8 in Fig.3), and then terminates activity A_i;

10) Server scheduler release resource (9 in Fig.2).

As we know, a real-time CORBA client uses activity as scheduling unit. An activity A_i is a set of remote methods: $M=\{m_1, m_2, \cdots, m_n\}$. To describe the cooperation process, several mappings are defined: $f_{client}(A_i)$, $f_{server}(m_i)$ and $f_{object}(m_i)$. $f_{client}(A_i)$ represents client scheduler that is responsible for scheduling activity A_i. $f_{server}(m_i)$represents the server scheduler that is responsible for scheduling the object in which method m_i is, $f_{object}(m_i)$ represents the server object. Activity A_i can be described as follows:

```
Get information about activity Ai, such as M={ m1, m2, ···, mn};
Get ClientScheduler reference;
//Enter try region
admit= ClientScheduler–>trySchedule(···);
IF (admit==FALSE)    THEN EXIT;    ENDIF
//Exit from try region, then enter operation region
//Start to invoke remote methods
...
//End to invoke remote methods
//Exit from operation region, then enter exit region
ClientScheduler–>Release(···);
//Exit, terminate Ai
```

Activity A_i can be divided into three regions: Try region, Operation region and Exit region. In Try region, client scheduler is called for admission test. If the test passes, activity A_i will enter Operation region, otherwise, aborts the activity. In Operation region, several remote invocations will be invoked in turn. In Exit region, resource will be released and activity A_i ends. It is necessary to point out that above operations are transparent to developer, only except that the operation in Operation region are defined by developer. For instance, a client can be written as follows:

```
ClientScheduler_var clt_sched= create scheduling service object;
object1_var obj1 = /* something */    //get and bind objects
object1_var obj1 = /* something */
BENGIN_ACTIVITY(clt_sched, “activity1”)
obj1 –> method1 ( ) ;
obj2 –> method1 ( ) ;
END_ACTIVITY
BENGIN_ACTIVITY(clt_sched, “activity2”)
obj1 –> method2 ( ) ;
obj2 –> method2 ( ) ;
END_ACTIVITY
```

From the above, it is clear that many operations are hidden by macro “BEGIN_ACTIVITY” and “END_ACTIVITY”.

Client scheduler mainly provides two operations for the scheduling activity: operation

"trySchedule" that makes admission test and operation "Release" that releases resource. They are defined as follows:

```
trySchedule(…)
{
FOR each mj∈Mi      { //request each Server Scheduler to make admission test
          ServerScheduler= fserver(mj)
          ServerScheduler->AdmissionTest(Req, …);
}
ENDFOR
Collect return value R={r1, r2, …, rn};
//If all return values are ACCETED, the test passes.
IF (∀ rj∈R, rj ==ACCEPTED ) THEN    return TRUE
ELSE     //Otherwis, release the resource that have been reserved
     FOR each mj∈Mi      {
          IF (rj== ACCEPTED) THEN {
               //request ServerScheduler to release resource
                ServerScheduler= fserver(mj)
               ServerScheduler–>Terminate( );}
          ENDIF
     ENDFOR
ENDIF
}
Release(…)
{
//request each Server Scheduler to release resource
FOR each mj∈Mi      {
          ServerScheduler= fserver(mj)
          ServerScheduler–>Terminate( );}
ENDFOR
}
```

In fact, a two-phase protocol is used in operation trySchedule. Only all servers can accept the corresponding request, then the activity will be scheduled, otherwise it will be given up.

Server scheduler mainly provides two operations: operation AdmissionTest that makes on-line schedulability analysis and operation Terminate that releases resource. They are defined as follows:

```
AdmissionTest(Req, …)
{
GET information from static repository and run-time repository;
make schedulability analysis;
IF Schedulable THEN {//If the server can accept it, it need to reserve corresponding resource
          Configure server and reserve resource;
          Store information about mi to runtime repository;
          Req.SendReply(ACCEPTED);
ELSE
          Req.SendReply(REJECTED);
ENDIF
}
Terminate( )
{
//Release resource
cancel server configuration and resource reserved;
```

delete information about m_i from run-time repository;

return;

}

The static repository and the run-time repository used in Server Scheduler will be defined in next section.

3.3 Dynamic CORBA System

The scheduling services implemented in the research can support both static scheduling and dynamic scheduling. A scheduler includes off-line scheduler and on-line scheduler. Based on the dynamic scheduling service, dynamic and open real-time CORBA system can be implemented as Fig.3.

The real-time applications and the configuration information are analyzed by off-line scheduler, and the corresponding static scheduling information is stored in static scheduling repository. The step is necessary for those applications that need fixed priority scheduling. But for the dynamic clients, on-line schedulability analysis should be done by on-line scheduler, in term of the information in dynamic information repository and the static scheduling information. The tasks will be dispatched by OS dispatcher according to the schedule that is created by scheduler.

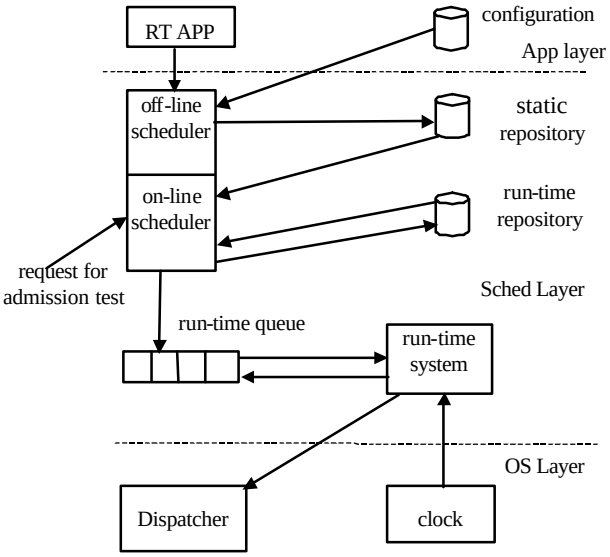

Fig. 3 Components of Dynamic CORBA System

4 Conclusion

Introducing scheduling service into real-time CORBA helps to simplify development of real-time applications. Scheduling service defined in real-time CORBA1.0 uses fixed priority scheduling methods, which only can be applied to a "closed" CORBA application system. In the paper, a dynamic scheduling service model is proposed, which can be applied to a dynamic, open real-time CORBA system. A dynamic client is permitted to execute on a particular processing node with simple admission test. The admission test is accomplished by cooperation of client scheduler and server scheduler.

References

1 Su Sen, Tang Xuefei, Liu Jinde. Object-oriented interoperability technology. Journal of University of Electronic Science and Technology of China, 1998, 27(1): 90~94[苏 森, 唐雪飞,刘锦德. 面向对象的互操作技术. 电子科技大学学报, 1998, 27(1): 90~94]

2 Object Management Group. real-time CORBA joint revised submission. OMG Document orbos/99-02-12 ed. 1999

3 Schmidt D C, Levine D, Mungee S. The design of the TAO real-time object request broker. Computer Communications Special Issue on Building Quality of Service into Distributed Systems, 1998, 21(4):69~86

4 Wolfe V F, DiPippo L C. Real-time CORBA. Proceedings of the Real-Time Technology and Applications Symposium, IEEE, 1997, 148-157

5 Feng W C, Syyid U, Liu W S. Providing for an open real-time CORBA. Proceedings of the IEEE Workshop on Middleware for Distributed Real-Time Systems and Services, IEEE. 1997,135~141.

6 Gill C, Levine D, Schmidt D C, *et al.* The design and performance of a real-time CORBA scheduling service. Real-time Systems, Kluwer, 2001, 20(2): 57~78

7 DiPippo L C, Wolfe V F. A scheduling service for a dynamic real-time CORBA system. in the proceedings of the twenty-second annual international computer software and application conference, IEEE, 1998,189~196

8 Deng Z, Liu W S. Scheduling real-time applications in an open environment. Proceedingsof IEEE 18th Real-Time Systems Symposium , IEEE, 1997, 308~319

9 Object Management Group. Dynamic scheduling initial submission. OMG Document orbos/99-10-06, 1999

10 Wang Zhiping, Xiong Guangze. Study of real-time scheduling algorithm. Journal of University of Electronic Science and Technology of China, 2000, 29(2):205~208 [王志平，熊光泽. 实时调度算法研究. 电子科技大学学报, 2000, 29(2):205~209]

11 Luo Lei, Xiong Guangze. Research on worst case design of real-time multitasking applications. Journal of University of Electronic Science and Technology of China, 1997, 26(1):74~77[罗 蕾，熊光泽.实时多任务应用最坏情况设计的研究. 电子科技大学学报，1997, 26(1):74~77]

一个实时 CORBA 的动态调度服务模型*

骆志刚** 谭 浩 刘锦德

(电子科技大学计算机科学与工程学院 成都 610054)

【摘要】 研究了实时 CORBA 调度服务的组成，并分析了它的局限性。利用客户调度器和服务器调度的协作，实现了新增应用的接纳测试。基于这种接纳测试方法，提出了一个动态调度服务模型，克服了实时 CORBA 调度服务的局限性，进而扩展了实时 CORBA 的应用范围。利用该模型，实现了一个动态、开放的实时 CORBA 系统。

关 键 词 实时 CORBA； 调度服务； 动态调度； 接纳测试

中图分类号 TP301.6

2001 年 8 月 27 日收稿

* 国防科研基金资助项目

** 男 27 岁 博士生