操作系统 时间服务 多线程 计算机 (23)

计算机科学1998Vol.25№ 6

99-102 TP316

时间服务的设计

Design of Time Service

旷海蓉 董歆奕 尤晋元

(上海交通大学计算机系 上海 200030)

摘 要 Time Service is one of the OMG Common Object Services. It enables the users to obtain current time together with an error estimate associated with it. This paper mainly talks about the design of time service, including how to synchronize time and how to generate time-based events based on timer and alarms.

关键字 Time Service, Synchronization, CORBA

1 引言

1.1 CORBA 对象服务

CORBA 是 OMG 组织制定的分布对象计算设计规范，其目的是在分布异构环境中实现信息和资源共享。对象服务是建立在 CORBA 核心 ORB 之上的服务接口，它独立于应用领域，能为许多对象程序所共有。这些对象服务提供如对象生命周期管理、永久对象、命名、属性等与分布对象本身相关的基础服务，也提供如事件服务、查询服务、安全服务、版本管理等更高层的复杂服务。本文所研究的时间服务正是其中之一。

1.2 时间服务

在集中式系统中时间是无二义性的，而在分布式系统中，不存在共同的时间或其它精确的时间来源。因此，时间同步是分布式系统的一个基本要求，只有取得一致的时间，各节点之间才能协调工作。

OMG 组织制定的时间服务分基础时间服务和定时事件服务。

基础时间服务提供取得当前时间及操纵时间的接口。该服务管理两类对象：通用时间对象及时间间隔对象。通过服务接口，用户可以获取当前时间及该时间与实际时间的误差估算，确定事件发生的次序，计算两个事件之间的间隔。

定时事件服务提供了定时触发分布系统中事件的服务。定时事件服务负责管理 TimerEventHandler 对象。用于记录在某个时间将要触发的事件数据及其事件触发的行为。

2 基础时间服务的设计与实现

本文设计的时间服务依赖于如下的假设：存在一个底层的时间服务器能提供合理的当前时间。该底层时间服务具有如下特点：

- 该时间服务能提供当前时间及其相关的时间误差参数。
- 该时间服务在一定条件下可以认为是有效和可靠的，而且它提供的时间是在该系统规定的精度误差内。
- 该时钟服务返回的时间是单调递增的。

2.1 时间服务的体系结构

图1描述了我们提供的时间服务的体系结构。它可为大学校园等大型单位提供时间服务。其中有三个分布时间服务，为校园网的计算机用户提供物理时间。第一层的可靠时间服务能为这些分布时间服务器提供与 WWWVB 等外部精确时间来源同步的时间。

高层的时间通过同步协议将精确时间向低层的时间服务器传递。同步算法设法使所有计算机的时间保持为同一 UTC 标准。越靠近顶层的时间服务器层次越小：如第一层是 UTC 时间的直接来源(广播、卫星等)；从第一层接收时间的时间服务器位于第二层；第三层服务器如院系时间服务器等从第二层取得时间。在层与层之间的计算机时间同步具有典型的主/从结构。设主服务器与外部精确时间来源的最大时间误差为 em，主从服务器的最大时间误差为 ms，则从服务器与精确时间来源的最大时间误差

为 εm＋ms，同一层间的从服务器之间的最大时间误差为2ms。由此可见层次越高，则时间服务器与外部精确时间来源的时间误差越大，而且同层之间的时间服务器之间也需进行时间同步。

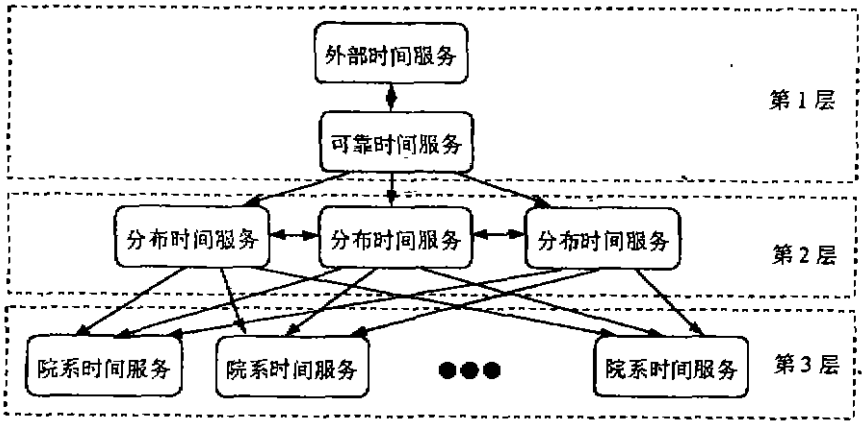

图1　时间服务的体系结构

上述体系结构中，第 k(k!=1)层一般存在多个时间服务器。第 k＋1层时间服务器与第 k 层服务器同步时，需要选择适当的服务器与之同步。主服务器的选择有多种方式。从服务器可以指定某一特定主服务器，也可由 ORB 通过多种策略为之选择一个：如任选一个、根据服务器负载选择最轻的一个，或基于安全性的考虑选择最合适的一个。总之，多种的选择策略提供了灵活性，也能为整个系统提供最好的性能(负载平衡、安全性好等)。

2.2　时间同步算法

因为时间同步对分布系统有着举足轻重的作用，从八十年代中期开始对时间同步算法的研究就从未间断，涌现了多种同步算法。这些算法主要分为三类：硬件、软件、及概率算法。硬件机制采用特殊的硬件设备提供非常精确的时间同步。软件机制一般为确定算法，假定同步消息在网络上传输的延迟存在上界。这类方法不用特殊硬件设备，而用较少的同步消息同步时钟，但是不提供物理时间。概率算法允许消息的网络延迟的随机性，需要大量的同步消息来获得所需精度的同步时间。这类算法不采用硬件设备，就能提供物理时间。

本文的工作在综合各种算法的基础上，针对图1中的体系结构，根据服务器之间不同的关系特点采用合适的同步算法。这些算法主要基于概率算法。

2.2.1　层间时间的同步算法　层间服务器时间的同步，具有明显的主/从结构，适合采用主从同步算法。为了与主服务器保持同步，每个从服务器必须周期性地试图去读取主服务器的时钟。每次成功读取又最多包含 k 次连续读取。若前次读取已经失败，从服务器再次读取。如果全部 k 次读取都失败，则离开这次同步组，等待下一次同步。关于层间时间同步算法的数学描述，见参考文献[2]。

2.2.2　层内时间的同步算法　层内计算机的特点是：各台计算机具有平等关系。与层间计算机的同步不同，不存在自然的主从关系。因此对于层内的时间同步算法我们虽然仍使用概率算法，但不采用主从形式。算法的思想如图2所示。层内 k 个计算机组成一个环。同步消息沿着时钟环流动，路径上的每个节点将本地时间加到消息中去。如同步消息在本地时间 T_0 从节点 n_0 出发，经过 k－1个节点，每个节点将本地时间 T_i 加到同步消息中，最后在本地时间 T_k 返回 n_0。每个消息在路径中传递两次，使得路径上的每个节点都能发送一次消息，后来又接收一次消息，取得消息中的所有时间戳。这样一个消息即可提供路径上所有时间的估算，从而大大减少了同步

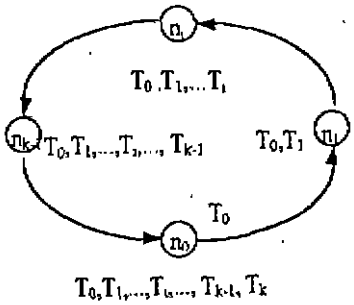

图2　层内时间同步消息路径

消息的数量。

算法根据消息中的信息对远地时间进行估算，估算的结果为一个时间间隔，包含远地时钟的当前值。层间同步的概率算法的思想是抛弃所有不满足精度的时间估算，直到满足精度的估算出现。与层间同步概率算法不同，层内同步算法保留每次时间估算，多次估算的时间间隔求交集，即可产生一个尽量小的时间间隔，满足同步精度的要求。因此大大加快了同步过程。关于层内时间同步算法的数学描述，见参考文献[2]。

无论是层间同步还是层内同步，当本地机获取远地时钟的估算后，要用软件实现调整本地机的逻辑时钟。然后逻辑时间仍以硬件时钟的速率运行，直到下一次成功读取远地时钟。

3 定时事件服务的设计与实现

3.1 定时事件服务方案的选择

定时服务是操作系统不可缺少的一部分，特别是周期性定时器更是操作系统的基本组成部分。定时服务在操作系统中有很多应用，比如定时服务可以支持进程的时间片抢占调度，从而支持传统的分时系统；定时服务还为进程提供其它与时间相关的服务，如在一天某个特定时间唤醒一个进程，或让进程睡眠一段时间；定时服务还能支持实时服务等。操作系统维持一个由硬件和软件支持的定时器队列。时钟中断服务程序周期性地进行处理，并对到时任务进行调度。

分布式系统是由多台计算机组成的，每台计算机依赖自己的硬件时钟具有自己的定时服务及定时队列。由于各台计算机时钟的差异，导致各台计算机下的定时触发不具有分布系统下的意义。所以在分布系统中，事件的触发时间应采用同步时间。

可选的解决方案有两种。一种是采用集中式管理方式，由于 OMG 组织提出的时间服务规范假定系统中存在一计算机与外部时钟保持同步，提供可靠的底层时间服务。该服务提供的时间可以认为是正确的物理时间。该计算机很自然可以作为系统中定时事件服务的节点，通过定时事件服务提供的接口设置的定时器，所有用户可以用系统调用将其映射为系统定时器，由操作系统统一管理。这一方案实现起来比较简单，但不能给用户提供可靠服务。一旦定时事件服务所在节点崩溃，则整个分布式系统不能提供定时服务。而且该方案还会给底层时间服务所在节点带来额外负载。该节点接收整个系统中所有的定时请求，并负责触发所有到时事件。

我们的实现仍采用分布管理的方法。定时事件服务可以分布在所有提供基础时间服务的计算机上。用户可以选择特定的服务器为之服务、也可以由 CORBA 随机选择或根据系统负载选择最合适的服务器。该方法使用基础时间服务提供的逻辑时钟，用软件实现定时服务，完全不依赖计算机的硬件时钟，因为基础时间服务提供的逻辑时钟能为分布系统中的所有事件正确排序，并提供相对高精度的物理时钟，所以定时事件服务提供的定时服务在分布系统中仍具有普遍的意义，触发的到时事件能在系统中有意义地传播。

3.2 定时队列的管理

定时事件服务维持两个队列：定时事件句柄对象队列和定时队列。定时事件句柄对象队列记录了所有注册的定时事件句柄。定时事件服务提供的 register() 操作可创建一个定时事件句柄对象、并插入到该队列中。unregister() 操作则将相应的定时事件句柄对象从该队列中删除。定时事件句柄对象记录了触发事件传送的目的地，传送的事件数据、上次设置的定时器分布触发时间、类型、状态等信息。这些信息除事件传送的目的地不可改变之外，其它都可动态改变。

一旦用户设置一有效触发时间，该定时事件句柄对象同时也被插入到定时队列中。定时队列是一个有序队列，以设置的事件触发时间先后为序。定时队列负责定时事件句柄对象的增删。定时事件服务支持三种类型的定时器：绝对时间定时器、相对时间定时器、及周期性定时器。绝对时间定时器要求系统在未来某个绝对时间触发相关事件；相对时间定时器要求系统在未来相对于设置时的时间的某个时刻触发相关事件；周期性定时器则要求系统周期性地触发某事件，若设置时的当前时间为 T，触发的时间间隔为 ΔT，则每次的触发时间依次为 T+ΔT，T+2 * ΔT，…等。

3.3 定时事件的触发

定时事件服务还负责到时事件的触发。它在初启时生成一触发线程负责定期检查定时队列是否有到时事件。由于定时队列是根据定时事件的触发时间先后排序的，触发线程只需从队首开始检查是否有到时事件。如果该事件已到时，则触发该事件，并将相应的定时事件句柄对象从队首移出，再继续检查下一个。一旦遇到某未到时事件或队列已为空，则中止这次检查。

定时事件触发的精度取决于触发线程检查的周期间隔的长短。假定该周期间隔的长度为 ΔT，意味着每隔 ΔT 时间触发线程将被唤醒，检查是否有到时事件。如果 ΔT 设置得较短，则事件触发的精度比较高，系统实时性比较强，到时事件能及时得到触发，但是如果 ΔT 设置得过短，则需频繁调度触发线程，增加系统负载，反而影响服务的性能。反之，如果 ΔT 设置得过大，则到时事件可能需要经过一段时间的延迟才能等到触发，得不到及时服务，从而影响定时事件服务的质量。由此可见，ΔT 的选择是影响定时事件服务性能的一个关键因素之一。

临界资源的管理也是定时事件服务设计中需考虑的问题。由于多线程的引入，定时队列及定时事件句柄对象均有可能被多个线程共享。但是如果定时队列在插入定时事件句柄对象的同时又实施删除操作，则可能导致队列无法保持有序等现象；如果在取消定时事件句柄对象的周期定时器的同时又触发该到时事件，则有可能导致该对象状态发生混乱。因此定时队列和定时事件句柄对象成为临界资源。一旦一个线程开始对其进行操作，则在该操作结束之前，其它线程就不能对它进行处理。目前实施临界资源互斥的方法有多种，如锁操作，信号量等。

由于多线程的引入和资源的共享与互斥，定时事件服务还可能引起死锁。当若干线程竞争使用资源时，如果每个线程都占有了一定资源，又申请使用已被另一线程占有，且不能抢占的资源，则所有这些线程都将进入封锁状态，不能继续运行下去，从而导致死锁。死锁的产生有多种条件，针对这些产生条件，存在相应的防止死锁的方法，如资源静态分配法，资源顺序使用法等，本文的工作采取资源顺序使用法，即给定时事件服务中的每一个临界资源分配一个唯一的编号，线程申请使用资源时，必须严格按照编号递增的次序进行。这样资源的分配是按序逐步进行，避免了资源分配请求图中循环链的存在，也就防止了死锁的产生。

结束语 基于以上设计思想，我们在 Visigenic 的 ORB 产品 Visibroker 上采用 Java 语言开发并实现了时间服务。在实现中，我们采用层次结构，通过层间时间同步和层内时间同步，取得全局时间同步，并根据层间和层内同步的不同特点分别采用主从式和时间环的概率同步算法，取得较高的同步精度，还基于全局同步时间实现了分布事件的定时服务。

参考文献

[1] OMG, CORBAservices: Common Object Service Specification, OMG Doc. No. 96-10-1, Chapter 14, Time Service Specification

[2] 旷海蓉，CORBA 环境下分布对象服务技术的研究，博士论文，上海交通大学计算机系，1997.12

[3] L. Lamport, Concurrent Reading and Writing of Clocks, ACM Trans. on Computer Systems, Vol. 8 Nov. 1990

[4] N. Vasanthavada, et al., Synchronization of Fault-tolerant Clocks in the Presence of Malicious Failures, IEEE Trans. on Comput., 37(4)1988

[5] Flaviu Cristian, Probabilistic Clock Synchronization, Distributed Computing, No 3, 1989

[6] Alan Olson, et al., Probabilistic Clock Synchronization in Large Distributed Systems, Proc. of the 11th Intl. Conf. on Distributed Computing systems, 1991

[7] Bulent Abali, et al., Clock Synchronization on a Multicomputer, Journal of Parallel and distributed Computing, No. 40, 1997

[8] D. L. Mills, Improved Algorithms for Synchronizing Computer Network Clocks, IEEE/ACM Trans. Networks, 3(3)1995

(上接第107页)

[4] A. Shatdal., et al., Using Shared Virtual Memory for Parallel Join Processing, Proc. of the 1993 ACM SIGMOD, Washington D. C., May, 1993

[5] J. L. Wolf, et al., An Effective Algorithm for Paralleling Hash Joins in the Presence of Data Skew, Proc. of the 1991 ICDE, 1991

[6] M. Kitsuregawa, et al., Parallel GRACE Hash Join on Share-Everything Multiprocessor: Implementation and Performance Evaluation on Symmetry S81, Proc. of the 8th ICDE, Temper, Arizona, Feb. 1992

[7] D. Schneider, et al., A Performance Evaluation of Four Parallel Algorithms in a Shared-nothing Multiprocessor Environment, Proc. of the 1989 ACM SIGMOD, 1989

[8] A. Shatdal, Architectural Considerations for Parallel Query Evaluation Algorithms, the Dissertation for the Degree of Doctor of Philosophy (Computer Sciences), University of Wisconsin-Madison, 1996