第 36 卷 第 2 期 计 算 机 工 程 2010 年 1 月
Vol.36 No.2 Computer Engineering January 2010
·软件技术与数据库· 文章编号：1000—3428(2010)02—0075—03 文献标识码：A 中图分类号：TP301.6

多线程应用中的定时器管理算法

姚崇华，姜新红，程凌宇，程永裕
(上海贝尔阿尔卡特股份有限公司，上海 200070)

摘 要：针对高性能电信系统中软定时器效率低下的问题，提出一系列优化方案，采用二次散列的时间轮，并结合免锁算法的低粒度互斥锁，从理论上把定时器查询和定时器插入等常用操作的复杂度从 $O(n)$降至最优情况的 $O(1)$。通过真实高负荷进行测试，采用 SunStudio11 性能分析工具对优化前后的性能进行定量分析。实验结果表明，该优化方案能够有效提高系统效率。

关键词：时间轮；二次散列；粒度；免锁算法

Timer Management Algorithm in Multi-thread Application

YAO Chong-hua, JIANG Xin-hong, CHENG Ling-yu, CHENG Yong-yu
(Alcatel Shanghai Bell, Ltd. Co., Shanghai 200070)

【Abstract】Aiming at the problem of low performance issue of soft timer manager which is commonly used in high performance telecom system, a series of optimized algorithm and solution are proposed including double hash time wheel and mutex with small granularity which is inspired by lock-free theory. It decreases the theoretical complexity from $O(n)$ to $O(1)$. Through high load test, and analysis is made with SunStudio11's performance analyzer, and the results show this optimized scheme can promote the system performance effectively.

【Key words】time wheel; double hash; granularity; lock-free algorithm

1 概述

在通信领域的产品(如大型核心交换网系统)中，有许多定时要求，有些任务需要周期性定时运行，如果主备用之间的数据同步，以保证一旦发生切换，备用设备能够马上接管任务，并且不存在不一致情况；有些任务需要延时，特别是在网络中，一方发送消息给另一方，通常需要等待延迟一段时间，此时需要定时器来提供定时服务，发送方并不能确定能够收到消息，这样就会无限等待下去，造成时间和资源的浪费，所以，通过定时器准确定时后，发送方可以进行相应超时处理，如重发或者放弃发送；还有一些任务需要等待其他事件或者消息，才能设置超时控制时间，如呼叫转移业务，在被叫方无应答一段时间后，呼叫转移处理被触发。总之，很多情况下都需要软件定时服务。

系统中所需的定时器数量是相当可观的，而且对系统的性能也有相当大的影响。如果定时器的工作效率较低，将会消耗大量的 CPU 资源[1]，不但会影响到定时精度，而且还会影响到系统的整体性能。因此，在系统中实现一个高效的定时器管理算法将对系统的稳定性以及系统的整体性能都有重要作用。

2 定时器管理算法介绍

对于一个通用的定时器管理算法来说，以下操作是必须向使用者提供的：

(1)添加定时器

addTimer(interval, timerId)：添加一个指定时长(interval)的定时器，timerId 用于唯一标识一个定时器。

(2)查找定时器

scanTimer(timerId)：根据 timerId 查找一个定时器。

(3)超时处理

timerTick：将定时器的时长划分为若干个时长相等的时间片，在每个时间片内检查是否有定时器超时。如果有，则触发超时处理，否则终止该定时器。对于需要长期运行的定时器，则不删除，需将其进行重置。

除了以上所描述的操作外，对于一个能够在多线程环境下稳定工作的定时器管理算法来说，还必须解决线程互斥和工作效率等问题，这些问题会在下文进行讨论。

定时器管理算法的性能指标有 2 种：(1)空间复杂度：对于该算法用于存储定时器集合的数据结构的内存空间的度量。(2)时间复杂度：该算法用于管理有限个定时器的最大执行时间的度量，它是直接反映该算法性能的指标[3]。

3 定时器管理算法原理

3.1 基于有序时间链表的定时器管理算法

(1)addTimer：遍历整个定时器链表，找到合适的插入点，将新定时器插入到链表中。时间复杂度为 $O(n)$。

(2)scanTimer：根据 timerId 在链表中查找定时器。时间复杂度为 $O(n)$。

(3)timerTick：添加定时器时，是按照定时器的超时时间间隔大小排序，所以，判断是否有超时的定时器只需看当前链表头部的定时器是否超时。时间复杂度为 $O(1)$。

图 1 显示了在当前系统使用的定时器的数据结构。

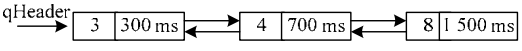

图 1 定时器链表结构

3.2 基于时间轮的定时器管理算法

从免锁算法着手，在多线程环境下，众多线程会使用同

作者简介：姚崇华(1978－)，男，高级工程师，主研方向：软交换平台；姜新红，高级工程师；程凌宇、程永裕，软件工程师

收稿日期：2009-10-15 **E-mail：**Lingyu.Cheng@alcatel-sbell.com.cn

一个定时器管理器，必须确保所有对这个共享的定时器管理器的、有可能导致竞争条件的操作都由互斥机制进行协调。改造前的定时器是使用链表进行管理，因此，每次对该链表的访问都要锁定整个数据结构。大粒度的锁对于其他线程而言，需要付出更多等待的时间，这势必影响多线程的并发性，进而影响系统效率。

免锁算法的思想是将共享数据，也就是互斥体本身划分成精细的小集合，对这些小集合所进行的操作是原子的。免锁算法编程带来的好处是在线程交互方面，借助于无锁算法，能够对线程的交互提供更好的保证。

然而，实现一个绝对免锁的算法是困难的，所谓“对这些小集合所进行的操作是原子的”的操作是怎样一种操作，这个问题相当于：最小需要一个怎样的集合才允许实现免锁算法[2]。

目前，对定时器管理器的改造集中于把一个大粒度锁变为多个小粒度锁来实现定时器管理器的“免锁”。

一个较大的链表被散列到一个数组中变为多个较小的链表，对这个链表管理器的访问所进行的互斥可以细化到对每个小链表进行。这样，多个线程就可以同时访问这个定时器管理器(如果想要访问的不是同一个小链表的话，大部分情况符合这个要求)。

简单链表的查找效率最低，时间复杂度为 $O(n)$，数组的查找效率最高，时间复杂度为 $O(1)$。而对于插入操作，正好相反。另外，由于定时器个数的不确定性，原来的算法采用基于链表实现，优点是简单、插入效率高，缺点是查找效率低下。这样的实现对于频繁需要在定时器超时之前取消定时的应用而言，效率极其低下。因此，为了保留链表的优点并且最大程度地弥补其缺陷，可以考虑结合数组和链表，使用组合数据结构来实现定时器管理器。基本的思想是：把大链表散列到一个固定长度的数组中。数组的长度为定时器管理器预计需要管理的定时器的最大个数，这样，在最好的情况下，数组的每个 slot 上挂接的链表长度为 1。这样一来就在插入和查找上达到一个较好的折中。即使最坏的情况，所有的定时器碰巧都挂接在数组的同一个 slot 上，就蜕化为优化之前的情况。但根据定时器启动时刻的不确定性可以认为，这样的情况极少发生。

针对定时器的典型操作对比原先的情况变为：

(1)addTimer：根据定时器的预期超时时间以及定时器管理器时针指向的当前 slot，可以定位到这个定时器将被挂接到的目标 slot，然后根据目标 slot 上的具体情况，将新定时器节点插入这个 slot 的小链表中，时间复杂度为 $O(1)$。

(2)scanTimer：根据计时器中记录的 slot 号直接定位到挂接在其上的小链表，从该小链表上查找目标定时器节点，时间复杂度平均为 $O(1)$，最差为 $O(n)$。

(3)timerTick：当定时器管理器时针将要指向下一个 slot 时，对这个 slot 上所有链表节点的圈数进行减一操作，如果结果等于 0，则表示该节点上的计时器超时。时间复杂度平均为 $O(1)$，最差为 $O(n)$。

3.3 改进后定时器管理算法

改进的数据结构如下：

(1)timeWheel：时间轮，实际上是一个循环数组，长度可定为应用所需的定时器的最大数量(maxTimers)，每个数组元素是一个链表。

(2)timerNode：定时器节点，该节点结构应包括的属性有：时间片号(slotNum)，时间轮圈数(round)。

(3)addTimer：根据定时器的超时时间(interval)和时间片的时长(timeslot)相除，得到该定时器超时所需的时间片总数(ticks)。根据该定时器所需的时间片总数(ticks)和当前的时间片号，得出该定时器在时间轮中应处的时间片号，如果该时间片尚为空节点，就把该定时器关联到这个时间片上；如果该时间片不为空，把该定时器插入该时间片上的定时器链表中，同时计算时间轮圈数。时间复杂度：最好情况为 $O(1)$；最差情况时所有的定时器都链接到同一个时间片上，时间复杂度为 $O(n)$。

```
ticks = interval / timeslot;
slotNum = currentSlot + (ticks % maxTimers);
round = ticks / maxTimers;
```

(4)scanTimer：根据给定的定时器的信息，找到其在时间轮中的位置。时间复杂度为 $O(1)$。

(5)timerTick：只要判断当前 slotNum 上的节点或者链表中的 round 是否为 0，就可以知道是否超时。时间复杂度为 $O(1)$。

图 2、图 3 给出了从一开始在时间轮中加入定时器，对当前时间片中定时器进行操作以及定时器到时的处理。为了方便解释，定义文中时间轮的时间片大小都为 100 ms。

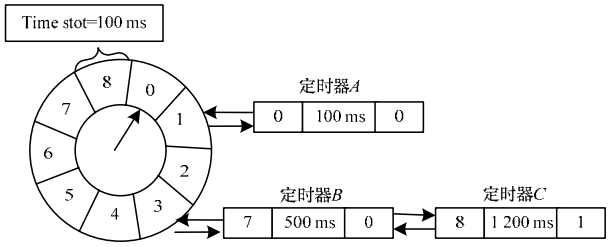

图 2　currentSlot=0 时，定时器 *A*,*B*,*C* 的添加

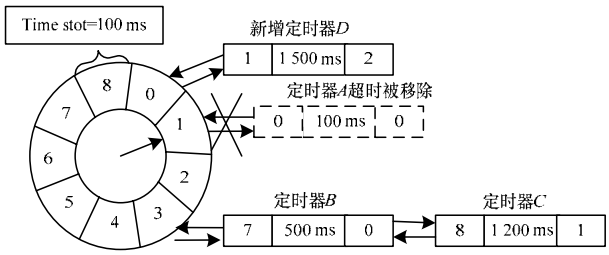

图 3　currentSlot=1 时，定时器 *A* 的移除及定时器 *D* 的添加

表 1 给出了改进前后 2 种算法的复杂度比较结果。

表 1　2 种算法的复杂度比较

algorithm	Time complexity	
	Add TimerNode	Scan for timer(by timer handle)
Old algorithm: Ordered Time List	$O(n)$	$O(n)$
New algorithm: Timing wheel	Average: $O(1)$ worst case: $O(n)$	Average: $O(1)$ worst case: $O(n)$

4　实验结果

系统中分别实现了这 2 种类型的定时器管理算法。运行环境为 Solaris10，使用 Sun Studio11 Performance Analyzer 分析 CPU 占用的情况。图 4、图 5 分别给出了测试数据统计结果，其中，使用原有定时器管理算法的统计结果为：addTimer 占用目标进程 CPU 的平均百分比为 4%；scanTimer 占用目标进程 CPU 的平均百分比为 2.6%。使用二次时间轮定时器管理算法统计结果为：addTimer 占用目标进程 CPU 的平均百分

比为 0.27%；scanTimer 占用目标进程 CPU 的平均百分比为 0.24%。

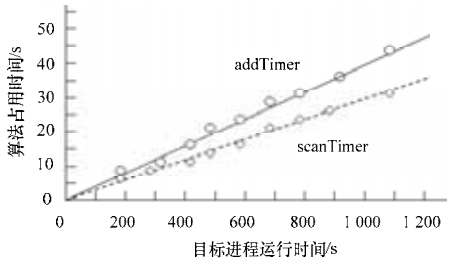

图 4　优化前算法占用目标进程的 CPU 时间

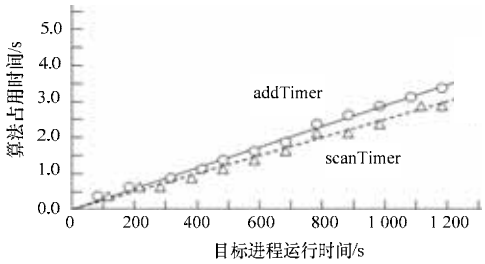

图 5　优化后算法占用目标进程的 CPU 时间

5　结束语

在本文的定时器管理算法中，一次散列操作将不同粒度大小的定时器用固定的时间片来划分；第 2 次散列根据应用允许的最大定时器个数 maxTimers，即时间轮的大小，将得到的 ticks 散列到时间轮上，从而把对原先的有序时间链表的查找操作变为根据定时器超时时间对应到时间轮上某个时间片的操作，时间复杂度由原来的 $O(n)$降低到 $O(1)$。最坏情况下，所有的定时器节点都被链接到时间轮的某个时间片上，时间复杂度为 $O(n)$。平均时间复杂度可以维持在 $O(1)$，同时在 Solaris10 上实现了这 2 种算法，使用 Sun Studio11 Performance Analyzer 工具对改进前后 2 种算法的性能数据进行收集和对比，实验结果表明，改进后的算法明显优于改进前算法。

参考文献

[1] Simon P. 30 Seconds Is Not Enough!: A Study of Operating System Timer Usage[C]//Proceedings of the 3rd ACM SIGOPS/EuroSys. European Conference on Computer Systems. [S. l.]: ACM Press, 2008.

[2] Varghese G. Hashed and Hierarchical Timing Wheels: Efficient Data Structures for Implementing a Timer Facility[J]. IEEE/ACM Transactions on Networking, 1997, 5(6): 824-834.

[3] 吴林平，胡仁杰，徐达银. 软件定时器的实现[J]. 工业控制计算机, 2002, 15(11): 46-48.

编辑　陈　文

(上接第 74 页)

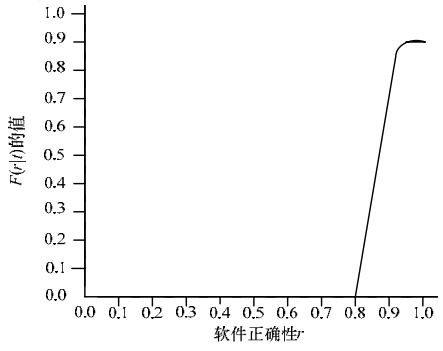

图 2　当 n=110,t=100 时 $F(r|t)$ 的取值

Matlab 平台可以获得 n=110，t=100 时 $F(r|t)$ 的函数自变量与函数值的对应结果，部分取值如下：

Columns 881 through 888

0.186 5　0.194 7　0.203 2　0.211 9　0.220 9　0.230 1　0.239 5　0.249 2

依据连续型随机变量概率分布的特性，可以得到：$F(r1) = p(r \le r1)$，则 $F(0.881)=P(r \le 0.881)=0.186\ 5$，且

$$P(r \ge 0.881)=1-0.186\ 5=0.813\ 5 \quad (10)$$

式(10)表示 n=110，t=100 时，软件正确性大于 0.881 的可靠性为 0.813 5，通过对比可知，之前求得的测试的可靠性符合式(10)的结果，即如果软件测试的可靠性要求达到 0.80，那么该中文学习平台的测试结果是可靠的。

该结果与笔者依据经验得到的结果是一致的，表明以上对软件可靠性的评估是可行的，且具有较强的理论性推理，有较强的科学依据和有很高的实用性。

5　结束语

软件的可靠性是衡量软件质量的一个重要指标。本文依据贝叶斯公式建立评估软件测试可靠性模型，根据经验值带入计算软件的测试可靠性。在假设已知软件正确性的情况下，程序的随机测试服从二项分布，可以通过假设确定软件正确性的先验分布下，数学推导出软件正确性的后验分布。具体的数值运算可以借助数学工具来实现(如 Matlab, Maple 等)，减少了繁琐的数学参数的确定过程，数形结合使得评估很直观，改进的基于贝叶斯理论的软件测试评估方法具有较好的实用性。

参考文献

[1] Paul C J. Software Testing: A Craftsman's Approach[M]. 2 版. 北京: 机械工业出版社, 2003.

[2] Keith W. Estimating Probability of Failure When Testing Reveal No Failures[J]. IEEE Transactions on Software Engineering, 1992, 18(1): 33-63.

[3] 张广梅. 软件测试与可靠性评估[D]. 北京: 中国科学院研究生院, 2006.

[4] 沈恒范. 概率论与数理统计教程[M]. 4 版. 北京: 高等教育出版社, 2003.

编辑　陈　文