面向实时系统的服务组件动态等级调度策略①

杨成群[1]，陈述平[2]，谭宏华[2]

[1](湖南涉外经济学院 计算机科学与技术学部，长沙 410205)
[2](湖南五强集团 艾因泰克科技股份有限公司，长沙 410205)

摘 要：OSGi 是基于组件的面向服务架构，其服务组件可在运行时远程地被安装、卸载和更新，但其服务实现的各项属性必须安装后才可知，这对于有时限要求的实时系统来说，是一个挑战。在执行时间服务和准入控制协议等研究基础上，提出了一种动态等级调度策略。通过动态地计算和分配组件的权限，以时间片轮转的方式调度组件，并设置不被中断的超级组件使服务组件能够应用在有特殊实时要求的系统中。最后通过实验和实践，分析和证明动态等级调度在面向服务架构上的良好应用。

关键词：SOA 架构; 实时系统; 服务; 调度; 组件;

Dynamic Hierarchical Scheduling Strategy of Service Components for Real-Time System

YANG Cheng-Qun[1], CHEN Shu-Ping[2], TAN Hong-Hua[2]

[1](College of Computer Science and Technology, Hunan International Economics University, Changsha 410205, China;)
[2](Hunan E-INTECH Technology Corporation, Wuqiang Corporation Group, Changsha 410205, China)

Abstract: OSGi is a service-oriented framework based on components, it is very useful to configurate dynamic reorganizable real-time system that the service components can remotely be installed, uninstalled and updated at run-time. But it is challenging the relation attributes are unknown until the services reach. On the basis of execution time service and admission control, this article proposed a dynamic hierarchical scheduling strategy (DHSS) which computs and assigns dynamically the rights of components and constituts a scheduling list. DHSS limited the running time of components by time slice circular scheduling and met the requirement of some especial real-time application by super-component which won't be interrupted. Finally experimental results and practical application analysis proves the feasibility of the DHSS on the SOA.

Key words: SOA architecture; real-time system; service; scheduling; components

1 引言

面向服务的架构（Service-Oriented Architecture，SOA）在各方面的研究和应用正在快速发展[1]，SOA 在很多领域的应用已经取得了成功，比如整合发展环境（IDEs）、家庭自动化产品、企业级应用系统、自动化工业等。目前，一个新的应用领域正在兴起，那就是实时系统[2]。很多的嵌入式系统是实时系统。这些嵌入式系统工作在复杂的环境，甚至是危险的环境中，需要应用配置能够动态可重构并可远程控制。例如，移动现场管理的系统有这样一些需求：系统必须对设备的运行状态进行实时的监测，例如设备运行的温度、振动加速度、转速等，这些值传送给外部的监控器。一旦传感器读到的值超出安全范围时，就需要执行远程控制，进行紧急处理来保证系统安全，如果确实需要，外部管理器应能完全停止设备的运转，以防止灾难性后果的发生。并且，当传感器硬件出现故障或者需要增加新型的传感器时，系统能够在运行时进行组件的替换和增加，而不需要关闭整个系统。

OSGi 框架是一种整合了面向服务概念和面向组件开发的组件框架[3]。面向服务的概念使其具有动态

① 收稿时间:2011-12-06;收到修改稿时间:2012-02-11

和可替换性，而不是一个静态的组件框架。面向服务的动态性是指允许组件在 OSGi 框架中运行时被安装、升级和卸载；可替换性是指允许服务在运行时被替换。在面向服务的系统中，服务请求被编译成一个服务接口，服务实现是未知的，也就是说，如果实现的执行时间是未知的，则服务的执行时间也是未知的。而且，服务组件的开发是独立完成的，开发者未必知道所有服务组件的时间属性。而实时系统中，每个组件都应有自己的时限，组件运行超过时限就会造成系统错误。例如，一旦传感器读到的值超出安全范围时，就需要执行关闭设备的服务，这样的紧急处理如果因为别的服务的运行而延误，将造成严重后果。所以，将 OSGi 应用于实时系统时，服务组件的时限控制就成了重要的问题。

OSGi 框架中的组件生命周期包括：安装、启动、更新、停止和卸载。服务组件的调度需要确保安装了新的组件后，系统的资源仍然能保证所有组件的运行[4]，这就需要一种准入控制。为了组件的可调度性分析，我们需要知道组件的服务周期。文献[5-6]提供了一种在线的方法来得到组件的 worst case execution time（最坏执行时间，WCET），即在最悲观情况下一个组件内所有实时线程的计算时间（C）、最小周期（T）、死线（或时限，D）。文献[7-8]提供了一种在线的 Response Time Analysis (RTA，回应时间分析) ，分析系统是否可调度。这些研究为本文的调度方法提供了基础。

在一些实时系统中还有特殊的实时要求，例如，设备实时故障监测系统中，用于停止设备运行的服务组件，如果该组件在时限到达后，也停止运行，这有可能造成严重后果。本文设计和实现的动态等级调度策略(DHSS, Dynamic Hierarchical Scheduling Strategy)，将动态地计算和分配组件的权限，为组件分配资源预算，以轮转的方式控制组件在时限内运行。并对有特殊实时要求的系统分配超级权限，保证组件运行时不被中断。

2 策略描述

动态等级调度的核心是：高等级的组件被优先调度，组件的等级由其拥有的权限决定。由于 OSGi 框架可以动态安装、卸载和更新组件，因此，DHSS 的关键之一就是实现权限的动态计算和分配。

2.1 动态权限计算

通过了可安装测试的组件被放入一个已安装组件列表,接下来的工作,就是计算每个组件的权限。对于服务请求者来说，从开始提供服务到完成服务的时间段称为组件服务的周期，在组件的调度中，它是影响调度效率的重要因素。权限计算的思想是：通过组件服务的周期来决定组件的权限，即服务周期较短的组件的权限较高，服务周期较长的组件的权限较低。组件中的线程可共享组件的权限，因此，组件中的线程将获得由组件服务周期决定的权限。组件的服务周期可由一系列的计算得到，详见文献[9]。下面通过表 1 中的例子来说明权限动态计算的规则。

表 1 组件权限计算

组件名称	服务周期（ms）	权限范围计算公式（r=权限总数/组件总数）	权限范围(min=0,max=63,r=16)	最终权限
C1	>3000	min+3r,max	48-63	55
C2	>2000 and <=3000	min+2r,min+(3r-1)	32-47	39
C3	>1000 and <=2000	min+r,min+(2r-1)	16-31	23
C4	<=1000	min,min+(r-1)	0-15	7

根据组件的服务周期，计算得到权限的级数（Grad,记为 G）。其中，第 i 个组件的服务周期记为 Pi。多个组件的权限构成一个集合 A，集合中的每一个值记为 Ai。

$$Ai=int(Pi/T) \quad (1)$$

$$G= counti(A) \quad (2)$$

公式(1)中的 T 是实时系统的时间级数，可根据实时系统的实际情况设置，例如 1000ms、100ms 或 10000ms。第 i 个服务组件的周期 Pi 除去时间级数后取整，得到第 i 个元素的时间常数值 Ai。例如，对表 1 中的组件进行计算，得到结果：A_1=int(3200/1000)=3，A_2=int(2200/1000)=2，A_3=int(1540/1000)=1，A_4=int(800/1000)=0。

公式(2)通过函数 counti()计算集合 A 中不同元素的个数。表 1 中组件构成的集合为：

A=|3，2，1，0|，则 Counti(A)=4，这就表示，当前框架中的组件等级 G=4。接下来，在权限总数 R 确定的情况下，通过公式(3)和公式(4)计算出第 n 级别的

权限范围。

$$r=R/G \quad (3)$$

$$min+(G-n)*r，min+n*r-1; \quad (4)$$

根据公式(3)计算出当前的权限常数; 根据公式(4)可计算出各级别的权限范围，如表 1 所示，当 R=64，G=4 时，则 r=16，min=0,max=63。所以，可计算出 C3 组件的权限范围为 16-31。

2.2 权限分配

服务组件的初始权限记为 w，可由公式计算得到。

$$w=(min+(G-n)*r+min+n*r-1)/2 \quad (5)$$

例如，C3 的权限范围是 16-31，则计算得到这一区间的中间值 23 作为 C3 的初始权限值。原因在于，以后安装的组件的周期比 C3 的周期长和比 C3 周期短的概率都是约 50%。据此，C1、C2、C3 和 C4 构成了权限值为 55、39、23、7 的链表。

这些组件根据它们所具有的权限形成一个从高到低的有序链表。当有新的组件要加入时，就可以直接插入到链表中的合适位置。

当然，还会存在一种情况，新加入的组件在要求的范围之内没有可用的权限供分配，例如，C3 的权限范围是[16，31]，如果后面又要加入了多个组件，用尽了 16-31 范围内的所有权限值，而新加入的组件 C20 的服务周期也在>1000 and <=2000 之内，此时，需要分两种情况来处理。一是新加入组件的服务周期与列表中的一个已经分配权限的组件相同，例如与 C9 相同，则 C20 将加入列表，成为 C9 的后件; 如果 C20 的服务周期与列表中组件不同，则需要重新计算所有组件的权限。

如前所述，在一些实时系统中，某些组件中的线程必须以最高权限继续执行，即使它已经超时了。为了满足这一需要，本文在前面组件权限范围计算的基础上，增加一个 super component（称为超级组件）的权限分配。超级组件的权限由系统指定为-1。所以，每个系统中的超级组件拥有最高的权限，位于最高的等级，一旦运行，就不会被任何其它组件的线程中断。

2.3 等级调度

组件安装后，加入到一个按照权限划分的调度列表中，组件的线程必须加入调度列表才能被调度。系统分配一个 CPU 时间预算/周期，相当于一个时间片。系统采用时间片轮转的方式调度组件。系统首先调度具有较高权限的组件，组件在预算/周期内才能运行，一旦组件的预算结束，则该组件的所有线程都停止运行，等待下一次调度。总之，要保证每个组件的线程只能使用不超过预算的时间。当组件启动后，组件中的线程将降低它们的权限至后台权限（即最低的权限），使得其它组件中的线程能够被调度; 在组件再一次被调度的开始，预算被重新填满，线程将它们的权限恢复至初始值。这就是本文实时服务组件调度的基本思想。

等级调度策略主要包括两部分：资源消耗监视和超时修复。通过“资源消耗监视”每个正在运行的组件的资源消耗情况，主要是 CPU 和内存，判断是否超出了预算。

如果组件超时（返回值为 false），则由“超时修复”部分执行以下操作：

① If (没有新组件加入调度列表)

② 组件权限=后台权限

③ 将组件加入到调度链表，成为链表中最后一个结点

④ 写入新的 CPU 消耗和预算值到 Real Time Schedule 文件

⑤ 激发一个异步中断事件，调度下一个结点

⑥ Else

⑦ 重新计算和分配组件权限

⑧ End if

2.4 关键问题

调度策略第一部分最重要的基础是对组件资源消耗的监测（Cost monitoring）。对每个线程的 CPU 执行时间进行监测，这是调度的基础。资源消耗采用轮询的方式来检测每个组件线程的标志，所以，这意味着需要一个对组件的所有线程进行管理的对象。

调度策略第二部分要求组件的线程是可异步中断的。当组件超时，应能安全地中断该组件的所有线程，而不是仅仅中断引起超时的线程。这样，才能保证在组件级别上进行调度。

DHSS 调度的约束是：D<=T ，即组件的时限小于或等于它的周期。因为，如果一个组件的时限大于它的周期，则不需要周期性的调度，这样的讨论也是没有意义的。另一个约束是：C<=D，即组件的预算小于或等于它的时限，若组件的工作一个周期内没有完成，则在下一周期内重填预算，继续完成工作。

3 仿真实验和分析

在这里，我们将服务组件集｛A_1,A_2,……，A_m｝映射为虚拟组件集Π={V_1，V_2，……，V_m}，每个虚拟组件用预算 Ci，时限 Di，周期 Ti 来描述。也就是说，每个虚拟组件定义为一个三元组（Ci，Di，Ti）。为了评估 DHSS 策略的性能，生成了多个不同的配置文件，用来表示具有不同时间要求的三元组，也就是待调度的虚拟组件，即周期范围为[10，10^2]、[10^2，10^3]、[10^3，10^4]（单位为 ms）等多个虚拟组件。

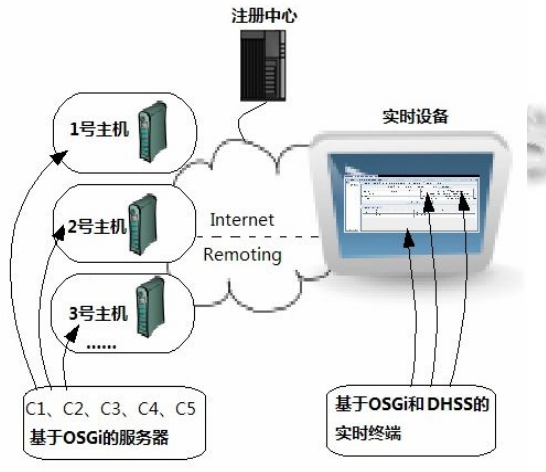

图 1 实验架构

实验在开放的网络环境下进行，实验架构如图 1 所示。实验首先实现了 4 个虚拟组件，这些服务组件位于不同的主机上，其服务周期、时限和预算如表 2 所示。

表 2 虚拟服务组件集

组件	周期（ms）	时限(ms)	预算（ms）	组件中的线程
C1	6450	5000	1000	T1 T2 T3
C2	4700	3100	800	T4 T5 T6
C3	2850	1500	500	T7 T8 T9
C4	1300	1020	200	T10 T11 T12

先在没有执行动态等级调度时，在终端依次安装和解析这四个组件，然后在执行 DHSS 时，重新安装和解析这四个组件。两种情况下组件解析的时间对比如图 2 所示。

为了得到策略在最坏情况下的效率，又实现了一个 C5（1300，1020，200），它的周期、时限和预算与 C4 相同，终端远程访问并申请 C5 的服务。由于每次调度时需要将 C5 插入到调度链表中作为 C4 的后件，可以从图3中看到C5被多次安装后每次解析时间的线性增长，所花费的时间可表示为：$T=a+k*b$，其中，a 表示安装每个组件时，基础服务所需要的时间，k 表示组件的个数（对应图 3 的横坐标）,b 表示每个组件执行 DHSS 需要的时间（对应图 3 的纵坐标）。

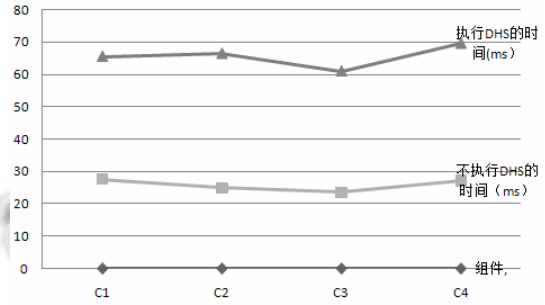

图 2 解析组件的时间对比

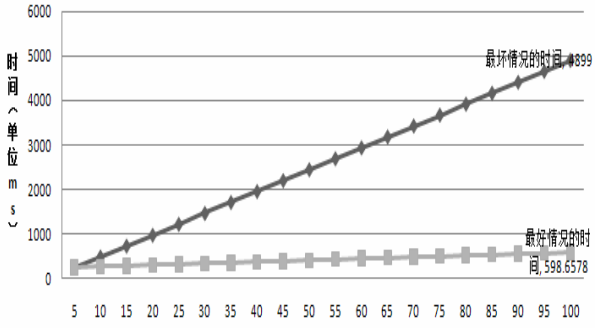

图 3 策略效率

经过数百次的重复运行，通过统计后计算，发现每个组件的调度时间平均约为 48.99ms。组件个数的增多，并不影响调度成功，但会延长调度时间。这是因为当服务组件的服务周期集中在一个等级内时，新增加组件时，由于权限分配不够，需要重新计算和划分权限而增加了时间。也就是说，为了执行动态等级调度，在组件解析阶段，需要对组件进行权限计算和分配，因此，组件的解析阶段所花费的时间与没有执行动态等级调度时相比，有约 45%的增长。虽然这个比率比较大，但这是因为解析花费的时间原本就比较小，所以执行 DHSS 策略增加了解析组件的比率就相对较明显了。最坏情况下，执行 DHSS 的时间仍然是一种线性增长，可表示为：$T=a+(k+\mu)*b$，其中，μ 是计算和分配权限所需要的时间系数，在该项实验平台上的值约为 7.6%。

4 DHSS在OSGi框架上的实现

OSGi 平台上的每个组件称为一个 bundle，bundle 如果控制了启动级别（Start Lever Service），那么也就可以控制整个服务平台。安装在实时终端上的 DHSS 服务在 OSGi 框架上通过 Service Permission [StartLevel,REGISTER]获得服务权限。

通过 DHSS 服务可对启动和停止 bundle 的顺序进行控制。DHSS 服务为每个 bundle 分配一个权限。DHSS 可以修改 bundle 的权限级别，并可以通过设置框架激活启动级别（active start level）来启动和停止相关的 bundle，实现对 bundle 的控制。DHSS 在 OSGi 框架上的实现如图 4 所示。

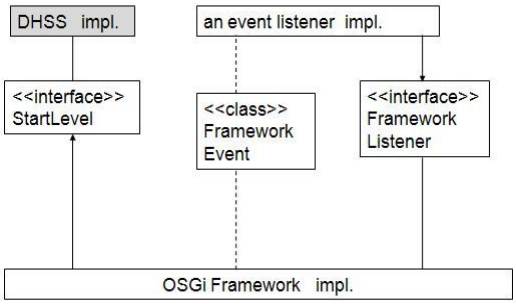

图 4　DHSS 在 OSGi 上的实现

DHSS 服务可以实现一种安全模式，在这种模式下，DHSS 服务本身的级别为 2，所有通过了可调度测试的组件的初始级别都为 1。DHSS 服务获得控制权，就开始对组件的权限计算和分配，构建一个待启动应用的链表。通过方法 BundleContext.getBundel()来构建这个按权限排列的链表。

方法 setInitialBundlestartLevel(int)设置 bundle 的启动级别初始值，该方法的整数参数就是 DHSS 服务所分配的权限值。然后，通过方法 setBundlestart Level(Bundle,int)来设置 bunlde 的启动顺序。系统 bundle 的启动级别为 0，这是不能被修改的。DHSS 服务可以通过调用启动级别服务 Start Lever Service 的方法：isBundlePersistentlyStarted(Bundle)，来检测每个标记为 1 的 bundle 是否已经按顺序启动。

可以通过 bundle 的 start()方法来启动 bundle，当服务组件的时间片到，用 stop()方法来停止 bundle 的运行，实现对服务组件的调度。

5 结语

面向实时响应系统的服务组件优化调度策略及策略，主要应满足其动态的实时资源分配和资源调度需求。实验结果表明，组件的服务周期与预算的大小是影响调度是否成功的关键因素之一。本文的 DHSS 调度现已应用于电力行业自适应智能移动现场管理系统中，实践表明，本调度方法能够实现服务组件的动态调度，较好地满足系统的实时性要求。

参考文献

1 吕建,马晓星,陶先平,等.网构软件的研究与进展.中国科学 E 辑信息科成本学,2006,36(10):1048−1054.

2 张仕,黄林鹏.基于 OSGi 的服务动态演化.软件学报,2008,19(5):1201−1211.

3 OSGi Alliance. OSGi service platform enterprise specification release 4. 2010,http://www.osgi.org/,March.

4 何秋生,涂时亮,陈章龙.OSGi 服务缓存的一种动态管理方法.小型微型计算机系统,2008,29(5):967−971.

5 Masson D, Midonnet S. RTSJ extensions:event manager and feasibility analyzer. Proceedings of the 6th international workshop on Java technologies for real-time and embedded systems. 2008,ACM:Santa Clara,California.

6 Davis RI, Zabos A, Burns A. Efficient exact schedulability tests for fixed priority real-time systems. IEEE Trans. Comput,2008,57(9):1261−1276.

7 Almeida L, Pederiras P. Scheduling within temporal partitions: response-time analysis and server design. Proceedings of the 4th ACM international conference on Embedded software.2004, ACM International Conference On Embedded Software: Pisa, Italy.

8 Strosnider JK, Lehoczky JP, Sha L. The deferrable deriver algorithm for enhanced aperiodic responsiveness in hard real-time environments. IEEE Trans. Compute., 1995.44 (1):73−91.

9 Zabos A, Davis RI, Burns A. Spare Capacity Distribution Using Exact Response-Time Analysis. http://www.cs.york.ac.uk/rts/papers_db_2009.php,2009.