数据中心实时交换系统的研究与实现

唐 旭[1] 王 飞[1] 李 彤[2] 张 鹏[1]
(西北核技术研究所数值模拟与软件研发中心 西安 710024)[1]
(清华大学计算机科学与技术系 北京 100084)[2]

摘 要 针对数据中心对信息实时交换系统实时可靠传输数据流的应用需求,分析了数据中心传输数据流的特点,设计了实时交换系统架构,重点研究了信息系统可靠、实时传输数据的方法。采用线程控制模块和消息同步阻塞 I/O 模型实现了并发数据流的处理,基于循环缓存机制和双系统同步技术解决了突发数据流易丢包的难题,保证了系统的可靠性。为了确保经过数据中心的数据流的实时传输,减少系统处理过程中产生的时延,提出了 QPTS(队列优先级驱动的任务调度)算法。该算法兼顾优先级、截止时间和剩余包数,实现了数据流的按需调度,提高了系统对交换数据流的处理速度。测试结果验证了算法的有效性以及系统的实时性和可靠性。

关键词 实时交换,循环缓存,优先级,截止时间,剩余包数
中图法分类号 TP302 **文献标识码** A

Research and Implementation of Real-time Exchange System in Data Center

TANG Xu[1] WANG Fei[1] LI Tong[2] ZHANG Peng[1]
(Center for Numerical Simulation and Software Engineering, Northwest Institute of Nuclear Technology, Xi'an 710024, China)[1]
(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)[2]

Abstract In view of real-time information exchange system(RIES) in data center, the applications require real-time and reliable transmission of data flow. This paper analyzed the features of data flow in data center, and proposed an architecture of RIES. Furthermore, this paper focused on the methods of reliable real-time data transmission. The methods uses the thread-control module and the message synchronous blocking I/O model to achieve the concurrent data flow processing. Moreover, the package loss of data flow is solved by using a loop caching mechanism, and the reliability of the system is ensured by double-system synchronized technique. In order to achieve real-time data transmission and decrease delay in the process of system processing, the queue priority based tasks scheduling (QPTS) algorithm was proposed. The algorithm implemented the on-demand scheduling of data flow, which is based on priority, deadline and amount of remaining packets, meanwhile it improved the processing speed of RIES. The evaluation results show the efficiency of the algorithm, and guarantee the real-time and reliability of the system.

Keywords Real-time exchange, Loop cache, Priority, Deadline, Amount of remaining packets

1 引言

近年来,数据中心在企业、院校、政府机构建设中逐渐普及,数据中心对信息实时交换系统的需求也越来越多。随着数据中心软硬件系统的不断扩建和功能的不断增加,中心之间的信息系统在运行环境、开发周期和技术集成等方面都存在着差异。为了解决"信息孤岛"的问题,以实现多个数据中心/设备之间的信息交互为目标,建成跨广域网的信息实时交换系统。因此,如何高效实现各信息系统间的数据流可靠、实时的交换已成为研究的热点和难点。

针对数据中心信息交换节点复杂、数据种类多、接口多、数据集成规模大、实时性和可靠要求高以及跨广域网完成信息交互的特点,本文采用多线程和同步阻塞 I/O 模型实现 1000 个节点以上并发信息的接收和分发;基于循环缓存和双系统同步技术实现信息的可靠传输,并优化系统资源。

另一方面,数据中心信息实时交换系统还应能够处理有时间限制的工作负载,实现数据流处理的实时性,保证尽可能多的任务在其截止期前完成,确保系统性能达到最佳[1]。由于调度决策涉及了大量的实时系统参数和实时任务属性,使得如何决策变得尤为困难[2]。

Liu 和 Layland[3] 首次提出了最早截止时间优先(Earliest Deadline First, EDF)任务调度算法,该算法根据任务截止时间决定任务的执行顺序,让截止时间最早的任务优先执行。Leung J[4] 和 Mok A 等人[5] 提出了最小空闲时间的优先算法(Least Slack Time First, LSF),该算法把作业剩余空闲时间作为优先级的评判标准,根据作业的剩余空闲时间动态分配。由于 LSF 算法的调度规则较为复杂,而且易发生颠簸,Hildebrandt 等人提出了增强的最小空闲时间的优先算法[6],缩短了算法中任务的运行时间。Jin Hong 等人[7] 提出了一种抢占阈值最小空闲时间的优先调度算法,降低了作业参数不确

唐 旭(1979—),女,硕士,高级工程师,主要研究方向为信息系统集成、实时系统、软件工程,E-mail:anabellat@126.com;王 飞(1981—),男,工程师,主要研究方向为计算机网络、数据库;李 彤(1989—),男,博士生,主要研究方向为网络体系结构、网络资源虚拟化。

定性对系统的影响。Semghouni 等人兼顾任务的重要性，提出了新的调度策略 GEDF[8]，提高了 EDF 算法处理系统过载的能力。王永炎等人[9]综合考虑任务的截止时间和价值两个特征参数的优先级表设计方法，提出了 EDV(earliest deadline value)与 VED(value earliest deadline)两种不同的基于优先级表的实时任务调度算法，该算法相对于 EDF 算法和 HVF 算法[10]有了很大的性能改进。

在数据实时交换系统中，如果一个实时任务在分配执行时间之前就已完成本周期内的工作，在这个空闲时间内其他实时任务也不能执行，则 CPU 处于相对空闲状态；如果一个实时任务超出了自己的执行时间，工作还没有结束，需要继续保持当前的高优先级继续执行下去，则会顺延本来已经安排的后续任务，从而形成“多米诺”效应，造成多个任务超出截止时间[11]。因此，只考虑截止时间(deadline)最早的数据流最优先，过载情况下无法保证所有的任务都能够满足截止时间，反而可能数据降低的处理效率。此外，时间紧迫的任务其价值未必高，而价值高的任务其执行时间未必紧迫[12]。因此，仅以参数 deadline 为标准无法满足实时处理的需求；而仅以空闲时间最短最优先(LSF)为参考标准，在过载情况下也会造成性能的降级[13]。由此可见，仅以某个特征参数来确定优先级是不够的[14-16]。因此，结合公平和效率两个方面，本文将队列优先级、数据流剩余包数(Amount of Remaining Packets)和 deadline 3 个特征参数作为评价标准，提出了一种基于队列优先级的任务算法(Queue Priority based Tasks Scheduling algorithm，QPTS)。首先，根据数据流优先级，将数据流加入对应的队列中；其次，兼顾队列优先级、剩余包数和 deadline 对每个队列中的数据流进行排序，设计了数据流调度算法；然后，通过对比 LSF 和 EDF 算法，验证了算法的有效性；最后，对数据中心实现的信息实时交换系统的性能进行了测试与验证。

2 系统框架

基于数据流的不同应用需求，信息实时交换系统应能够同时接收不同路由、不同链路传输的数据流。实时交互的信息主要有以下几个特点：1)部分应用的数据流的实时性要求高，需要尽早地被执行，部分数据流的实时性要求一般，被执行的紧迫性较低。2)数据包大小和发送频率不固定。根据发送数据频度将数据划分为两类，一类是基于数据流发送的数据，这种数据流没有固定的发送频率，数据包大小不固定，不仅容易造成网络拥塞，还容易产生丢包现象；另一类是基于速率发送的数据，这种数据流包大小和频率一般是固定的。3)信息节点多，较为分散，地理位置不固定。

数据中心实时交换系统作为与其他中心/设备以及中心内部各系统、设备之间信息交互的核心枢纽，为了保证信息实时、稳定、可靠地送达至目的地，本文基于双机模式设计了系统模型，如图 1 所示。模型包括多线程控制模块、信息操作控制模块和设备状态监测模块。此模型的主要思想是通过多线程、循环缓存、任务调度算法和设备状态监测等控制信息交换系统实现多种不同源、不同数据种类的信息交互。通过线程控制模块控制信息的采集和发送，在信息操作控制模块中采用循环缓存机制实现数据的容错处理，研究调度算法实现数据流的实时调度，通过设备状态监测模块实时监测主机和副机的运行状态，并通知多线程控制模块是否需要发送信息。

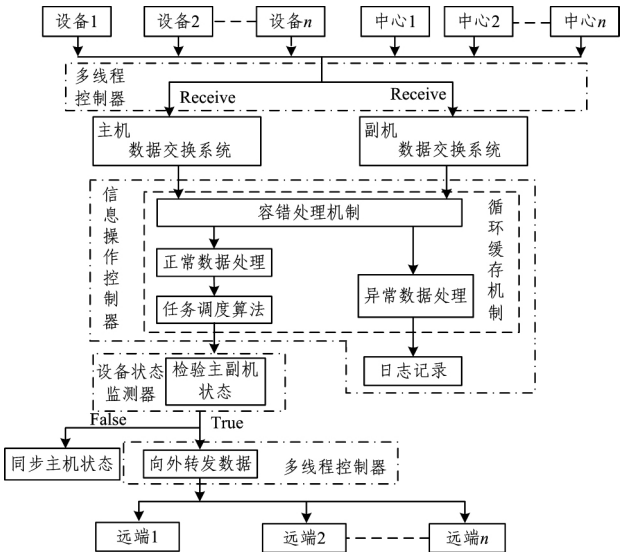

图 1 实时交换系统模型

信息实时交换系统主要实现以下功能：1)高频度实时信息的接收与分发，支持多个节点(1000 个以上)信息端的接入，并发交换数据；2)实现信息的容错、可靠处理，降低信息丢失率；3)针对不同优先级的数据流，确保重要数据流优先处理，实现数据低时延传输。根据实时交换系统模型和功能要求，本文设计了系统架构，通过配置文件定义了参数、变量，如信息交互的 IP 地址、端口号、信息类别等，以提高系统的灵活性；采用多线程和阻塞模型，实现并发信息的接入与分发；采用循环缓存技术、容错处理技术和主副机状态切换技术，确保该系统能够正确处理两种转发机制的数据流；同时基于信息的优先级，设计了优先级驱动的任务调度算法，实现了重要信息优先、实时传输至所需用户，降低了信息传输的时延。信息实时交换系统的架构设计如图 2 所示。

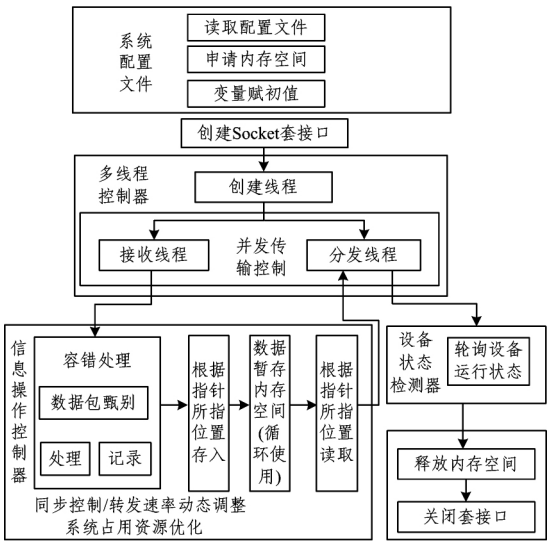

图 2 信息实时交换系统的架构

3 关键技术

3.1 并发信息接收与分发的设计

针对数据流特点，使用多线程控制模块实现信息的接收和分发，根据信息的信源和信宿建立线程，一个线程控制一类数据的交互。如数据采集由数据采集线程控制，数据发送由

数据发送线程控制。采用多线程的好处是可以提高 CPU 的利用率，占用更少的系统资源，从而提高系统工作的效率。然而线程也需要占用内存，线程越多占用内存也越多，而线程太多会导致控制太复杂，使容错性差。因此本系统对线程数量进行了控制，将流量较少的信源和信宿进行合并，对达到 10MB/s 以上速度的数据流建立独立线程，从而改善系统的结构和性能。

另一方面，采用基于消息的同步阻塞 I/O 模型，该模型基于主动请求并等待 I/O 操作完成，当数据就绪后在读写时阻塞，只有等待要操作的数据准备好，将数据从系统缓冲区复制到用户空间，然后该函数返回[17]。针对不同的网络事件进行登记，一旦有数据到来，就触发这个事件，操作系统则通过消息通知调用线程，后者就可以在相应的消息响应函数中接收到这个数据。

3.2　系统容错、可靠性的设计

系统基于循环缓存技术，申请内存空间存储接收到的数据，该空间与接收、转发线程唯一对应，并循环使用，其大小取决于数据频率和转发速度。采用容错技术，对缓存中的数据进行甄别，剔除异常数据并进行记录。其优点是：1）提高系统接收数据的可靠性，尤其是接收突发的大流量数据；2）减小转发线程处理数据的压力；3）转发线程可根据缓存空间内未处理的数据量动态调整处理速度，为优化系统资源占用提供了可能。循环缓存的设计如图 3 所示。

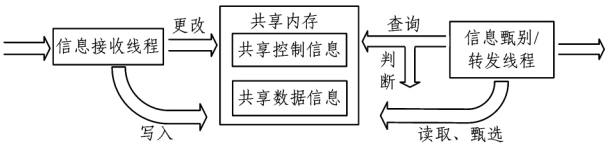

图 3　循环缓存的设计图

为进一步提高整个系统对数据处理的可靠性，采用双系统同步技术，同时运行主副机上的信息实时交换系统，该系统运行在两台 SUN 服务器上，实现双机热备。主副机上的实时交换系统同时接收数据，设备状态监测模块侦听主副机运行状态，只有主机上的系统才向外发送数据。当主机发生故障不能正常工作时，副机实时交换系统则立即向外转发数据。这样做的好处是保证了经过系统处理的数据包无丢失地传输，减小了因服务器故障或单方网络故障造成的数据损失。

3.3　优先级驱动的任务调度算法的设计

为了减少数据流在队列中的处理时间，本文综合优先级、剩余流包数和截止时间 3 个参数，设计了一套 QPTS 算法。该算法主要分为 3 个步骤来实现：

步骤 1　假设有 $N(N>1)$ 个队列（*queue*），定义第 i 个队列的优先级为 L，其中，$i\in[1,2,3,\cdots,N]$，$L\in[1,2,\cdots,N]$，优先级 L 为 1 的任务表示具有最高优先级。数据流根据其应用实时性要求定义属于第几个队列，且具有相同实时性要求的数据流归为同一队列。定义数据流的优先级权重为 ω。

$$\omega=\frac{1}{2^L} \tag{1}$$

从式(1)中可以看出，优先级越高，该数据流的权重因子也就越大。

基于优先级的队列中有不同的数据流（flow）在等待处理，高优先级的数据流会先被处理，只有当高优先级队列中没有数据流时，才处理下一个优先级队列中的数据流。例如，5 个队列的优先级 $L\in[1,2,3,4,5]$，每个队列中都有在等待处理的数据流，那么系统会首先处理 L 为 1 的队列中的数据流，等待该队列中数据流处理完后，才处理 L 为 2 的队列中的数据流，依次执行。如果前一优先级队列中没有等待处理的数据流，系统会自动在下一个优先级队列中发现需要处理的数据流，以此类推。

步骤 2　假设数据流处理完成时间（Completion Time）为 C，截止时间（deadline）为 d，flow 剩余数据包的数量为 r，数据包为 P，那么对于某个流的第 i 个包，其定义为：

$$\rho_{P_i}=\alpha\cdot Normalize(d_i)+Normalize(r_i) \tag{2}$$

其中，α 是一个实时性指标，表示系统对 deadline 的敏感程度。采用归一函数对参数 d 和 r 进行归一化处理，ρ_{P_i} 越小，P_i 会越早被优先服务。

其中，归一函数 Normalize 定义为：

$$Normalize(x)=\frac{x-x_{\min}}{x_{\max}-x_{\min}} \tag{3}$$

定义 1　M 为数据流 flow 的集合，对于 flow 的一个给定集 $m\in M$，不失一般性，调度后重新排列的流序列为：$A=\{f_1,f_2,\cdots,f_{|M|}\}$。

步骤 3　假设包处理时间为常数 t，系统只对数据包头进行解析（数据包包头格式如表 1 所列），因此不需要考虑传输数据包的大小，可以近似认为不同流的数据包的处理时间相等。C_m 表示第 m 个 flow 的完成时间，Q_m 为等待时间，则有：

$$C_m=N_m\cdot\Delta t+Q_m \tag{4}$$

其中，

$$Q_m=\begin{cases}0, & m=1\\ \sum_{i=1}^{m-1}r_i\Delta t, & m\in(1,M]\end{cases} \tag{5}$$

表 1　数据包包头格式

序号	名称	字节数
1	版本	2
2	数据长度	2
3	日期	6
4	数据名称	4

定义 2　d_m 表示第 m 个 flow 的 deadline，根据文献[18]定义第 m 个 flow 的延迟时间 Tardniss 为 T_m：

$$T_m=\max\{0,C_m-d_m\} \tag{6}$$

则优化目标函数值可以表示为：

$$Z=\sum_{m\in M}\omega T_m \tag{7}$$

Z 越小，说明该算法越优。

算法说明：根据系统接收的数据流的特点，仅仅以截止期（deadline）作为调度参数，有些同等优先级的数据流不能在最短时间内到达，会增加处理时延。例如，同一队列中同时有两个数据流 f_1 和 f_2，当 f_2 在调度时，f_1 到达，假设 f_1 和 f_2 的 $d_1=50$，$N_1=N_2=50$，$r_1=50$，$d_2=70$，$r_2=1$。采用 EDF 算法，会先处理 f_1，再处理 f_2，那么 $C_1=50$，$C_2=100$，根据式(5)和式(6)，$Z=T_1+T_2=30$。如果先处理剩余包更小的 f_2，再处理 f_1，$C_1=50+1$，$C_2=50$，则 $Z=T_1+T_2=1$。因此在这种情况下，以剩余最小包数为参考标准，反而能够得到更优解。本文综合考虑优先级、剩余数据包的个数（amount of remaining data packets）和 deadline 3 个特征参数作为数据流

处理的标准，提出了一种优先级驱动的 QPTS 任务调度算法，提高了系统的性能。

为了确保该算法的有效性，本文对算法进行了验证。假设有 10 个数据流($M=10$)和 5 个队列($L\in[1,5]$)，取 $d_m\in[1000,3000]$(所有时间单位为 us)，$r_m\in[1,1000]$，$t=1$，$\alpha=0.5$，算法运行的结果如图 4 和图 5 所示。

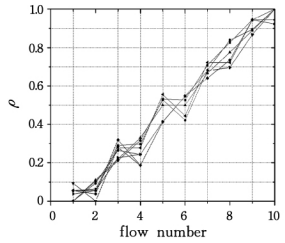

图 4 ρ_{P_i} 的计算结果

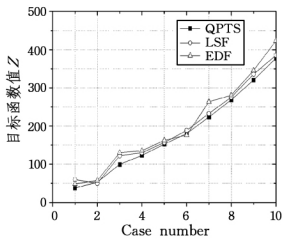

图 5 3 种算法目标函数值的比较

图 4 为 10 个数据流调度的结果，其中，X 轴表示被测数据流序号。可以看出，按照 QPTS 调度算法可以得到数据流调度的顺序，10 个数据流能根据其优先级、剩余数据包数和 deadline 合理调度，而不会受到某个单一参数的影响。图 5 表示在 10 种不同的情况下将 QPTS 算法与 LSF 和 EDF 算法进行比较，其中，X 轴表示测试实例。可以看出，QPTS 算法在大多数情况下的目标函数值都小于 LSF 和 EDF 算法，这主要是因为 LSF 算法仅仅依据剩余最小空闲时间进行优先级排队，对数据流的截止时间和队列优先级不敏感，而 EDF 算法则把截止时间作为优先排队的依据，对数据流剩余包数和截止时间不敏感，所以基于优先级队列的数据流 QPTS 算法的性能要优于 LSF 和 EDF 算法的性能。

4 系统测试与应用

基于标准 ANSI C 语言，采用 UDP 协议实现了上述系统的原型，系统是运行在 SUN 服务器 UNIX 系统上的应用软件，配置为 CPU 主频 2.6GHz，8 核，内存为 128GB，磁盘为 2 个 292GB 的 SAS 盘。

实验 1 采用 8 台计算机模拟信息发送站点对系统进行测试，以验证系统交换数据的整体性能。测试结果如图 6 所示。

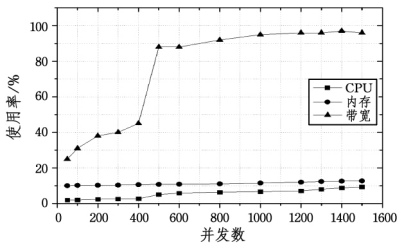

图 6 性能测试结果

图 6 中传输测试的每个数据包的大小都为 100kB。从图中可以看出，系统占用的 CPU 和内存都小于 20%，当并发数大于 500 时，系统占用带宽急剧增大，这主要是因为并发数的增加致使系统对 CPU 使用需求增大，带宽最大使用率为 97%，在保证不丢包的情况下系统最大可交换每秒 60MB 的数据。

实验 2 模拟两种类型的数据流，测试 a 模拟[1,200]Hz 之间固定频率的数据流，每组数据流的大小为 100kbyte；测试 b 模拟非固定频率的数据流，测试数据流大小为 1Mbyte，分别测试经过系统的时延大小。测试结果如图 7 和图 8 所示。

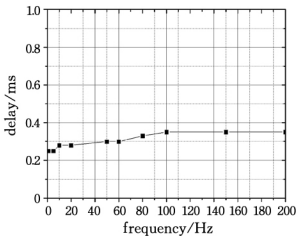

图 7 固定频率数据流测试结果

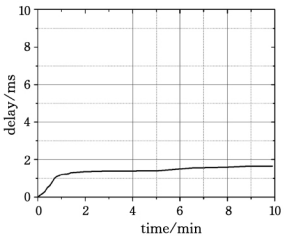

图 8 非固定频率数据流测试结果

从图 7、图 8 测试结果可以看出，当系统接收到固定频率的数据流时，系统对数据的处理时延基本不会受频率大小的影响；当系统接收到瞬时数据流时，数据自动暂存在缓存里，因此传输时延会逐渐增大，当缓存里未发数据过多时，系统会自动调节转发频率，所以时延逐渐趋于稳定。

结束语 本文研究并设计了数据中心实时交换系统，分析了数据中心传输信息的特点，设计了系统的模型与架构，采用多线程控制模块和基于消息的同步阻塞 I/O 模型实现了并发信息的采集与发送，使用循环缓存技术解决了大小和频率不固定的数据包易丢失的问题，同时采用双系统同步技术提高了系统的可靠性，最大限度地保证了信息实时、可靠传输至远端目的地。综合考虑不同队列数据流的优先级、deadline 和剩余包数，设计了优先级驱动的 QPTS 任务调度算法，实现了系统对数据流的实时处理，通过实验验证了该算法的有效性。通过对系统整体性能的实际测量，验证了系统的实时性和可靠性。该系统可以实现分布在不同地理位置的应用系统间的信息的实时交互，满足了远端接收节点对系统传输的实时性和可靠性要求，通过实验验证了该系统的性能。系统从网络层层面优化了数据流的调度，考虑了数据流在服务器上的处理时间。由于篇幅问题，没有涉及网路环境下的实时数据时延、网络负载等对系统造成的影响分析，这些将在后续的工作中开展进一步研究。

参 考 文 献

[1] HARITSA JAYANT R, CEREY MICHAEL J, LIVNY M. Value-based scheduling in real-time database systems[J]. VLDB Journal, 1993, 2(2): 117-152.

[2] ALDARMI S A, BURNS A. Dynamic value-density for scheduling real-time systems[C]// Proceedings of the 11th Euromicro Conference on Real-Time Systems, 1999. York: England, UK, 1999: 270-277.

[3] LIU C L, LAYLAND JAMES W. Scheduling algorithms for multiprogramming in a hard real-time environment[J]. Journal of the ACM, 1973, 20(1): 46-61.

[4] LEUNG J, WHITEHEAD J. On the complexity of fixed-priority scheduling of periodic real-time tasks[J]. Perfermance Evaluation, 1982, 2(4): 237-250.

[5] MOK A. Fundamental design problems of distributed systems for the hard real-time environment [D]. Cambridge: Massachusetts Institute of Technology, 1983.

[6] HILDEBRANDT J, GOLATOWSKI F, TIMMEMANN D. Scheduling of hard-real-time tasks[J]. IEEE Trans on Software Engineering, 1989, 15(12): 1497-1506.

(下转第 490 页)

参 考 文 献

[1] KHATOON S, MAHMOOD A, LI G. An evaluation of source code mining techniques[C]// International Conference on Fuzzy Systems and Knowledge Discovery. 2011:1929-1933.

[2] PICCIONI M, FURIA C A, MEYER B. An Empirical Study of API Usability[C]// Empirical Software Engineering and Measurement. New York: ACM, 2013:35-44.

[3] ROBILLARD M P. What makes apis hard to learn? Answers from developers[J]. IEEE Software, 2009, 26(6):27-34.

[4] THUMMALAPENTA S, XIE V. PARSEWeb: a programmer assistant for reusing open source code on the Web[C]// Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering. New York: ACM, 2007:204-213.

[5] LI Z, ZHOU Y. PR-Miner: automatically extractingimplicit programming rules and detecting violations in large software code [C]// European Software Engineering Conference/Foundations of Software Engineering. New York: ACM, 2005:306-315.

[6] XIE T, PEI J. MAPO: Mining API usages from open source repositories[C]// Proceedings of the 2006 international workshop on Mining software repositories. New York: ACM, 2006:54-57.

[7] NGUYEN T T, NGUYEN H A, PHAM N H, et al. Graph-based Mining of Multiple Object Usage Patterns[C]// European Software Engineering Conference/Foundations of Software Engineering. ACM, 2009:383-392.

[8] AKBAR R J, OMORI T, MARUYAMA K. Mining API Usage Patterns by Applying Method Categorization to Improve Code Completion[J]. IEICE Transactions on Information and Systems, 2014, 97(5):1069-1083.

[9] SAIED M A, BENOMA R O, SAHRAOU H, et al. Mining multi-level API usage patterns[C]// 2015 IEEE 22nd International Confe-rence on Software Analysis. 2015:23-32.

[10] 廖兴,尹俊文,蔡放. 基于 Java 语言的抽象语法树的创建与遍历[J]. 长沙大学学报, 2004, 18(4):50-53.

[11] 孙吉贵,刘杰,赵连雨. 聚类算法研究[J]. 软件学报, 2008, 19(1):48-61.

[12] ACHARYA M, XIE T, PEI J. Mining API patterns as partial orders from source code: from usage scenarios to specification [C]// European Software Engineering Conference/Foundations of Software Engineering. New York: ACM, 2007:25-34.

[13] CASAS-GARRIGA G. Summarizing Sequential Data with Closed Partial Orders[C]// 5th SIAM International Conference on Data Mining. 2005.

[14] WANG J, XIE T, ZHANG D, et al. Mining succinct and high-coverage api usage patterns from source code[C]// Working Conference on Mining Software Repositories. 2013:319-328.

[15] PEI J, WANG H, YU P, et al. Discovering frequent closed partial orders from strings[J]. IEEE Transactions on Knowledge and Data Engineering, 2006, 18(11):1467-1481.

[16] ZHONG H, XIE T, PEI J, et al. MAPO: mining and recommending API usage patterns[C]// the 23rd European Conference on ECOOP. 2009:318-343.

[17] ZAKI M. SPADE: An Efficient Algorithm for Mining Frequent Sequences[J]. Machine Learning, 2001, 42(1):31-60.

[18] WANG J, HAN J. BIDE: efficient mining of frequent closed sequences[C]// 20th International Conference on Data Engineering. 2004:79-91.

[19] MICHAIL A. Data mining library reuse patterns using generalized association rules[C]// Proceedings of the 22nd International Conference on Software Engineering. 2000:167-176.

[20] SAHAVECHAPHAN N, CLAYPOOL K. XSnippet: mining For sample code[J]. ACM SIGPLAN Notices, 2006, 41(10):413-430.

[21] HSU S K, LIN S J. MACs: Mining API code snippets for code reuse[J]. Expert Systems with Applications, 2011, 38(6):7291-7301.

(上接第 462 页)

[7] 金宏,王宏安,王强,等. 改进的最小空闲时间优先调度算法[J]. 软件学报, 2004, 15(8):1116-1123.

[8] Semghouni S, Amanton L, Sade B, et al. On new scheduling policy for the improvement of firm RTDBSs performance[J]. Data & Knowledge Engineering, 2007, 63(2):414-432.

[9] 王永炎,王强,王宏安,等. 基于优先级表的实时调度算法及其实现[J]. 软件学报, 2004, 15(3):360-370.

[10] 夏家莉,陈辉,杨兵. 一种动态优先级实时任务调度算法[J]. 计算机学报, 2012, 35(12):2686-2695.

[11] MARCO C, GIORGIO B, LUI S. Handling execution overruns in hard real-time control systems[J]. IEEE Transactions on Computer, 2002, 51(7):110-118.

[12] JENSEN E D, LOCKE C D, TODUDA H. A time-driven scheduling model for real-time operating systems[C]// the 6th IEEE Real-time System Symp, 1985. San Diego: IEEE Computer Society Press, 1985:112-122.

[13] BUTTAZZO G, SPURI M, SENSINI F. Value vs. Deadline scheduling in overload conditions[C]// the 19th IEEE Real-Time System Symp, 1995. Pisa: IEEE Computer Society Press, 1995: 90-99.

[14] BIYABANI S R, STANKOVIC J A, RAMAMRITHAM K. The integration of deadline and criticalness in hard real-time scheduling[C]// the 9th IEEE Real-Time System Symp, 1988. Huntsville: IEEE Computer Society Press, 1988:152-160.

[15] TSENG S M, CHIN Y H, YANG W P. Scheduling value-based transactions in real-time main-memory databases[M]// LIN K J, ed. the First International Workshop on Real-Time Databases: Issues and Applications. Newport Beach: Kluwer Academic Publishers, 1996:111-117.

[16] BURNS A, PRASAD D, BONDAVALLI A, et al. The meaning and role of value in scheduling flexible real-time systems[J]. Journal of Systems Architecture, 2000, 46(4):305-325.

[17] 胥光辉,徐永森. 同步阻塞线程的唤醒问题研究[J]. 计算机科学, 2002, 29(12):49-50.

[18] LI T, XU K, SHEN M, et al. Towards Minimal Tardiness of Data-intensive Applications in Heterogeneous Networks [C]// IEEE International Conference on Computer Communication and Networks (ICCCN), 2016. Hawaii: IEEE Computer Society Press, 2016:1-9.