第37卷 第18期
Vol.37 No.18

计 算 机 工 程
Computer Engineering

2011年9月
September 2011

·软件技术与数据库·

文章编号：1000—3428(2011)18—0028—04 文献标识码：A 中图分类号：TP311.5

低延迟证券交易系统关键技术研究

徐广斌[1,2]，武剑锋[1]，白 硕[1]

(1. 上海证券交易所技术中心，上海 200120；
2. 复旦大学计算机科学技术学院，上海 200443)

摘 要：为构建低延迟证券交易系统，给出对证券交易延迟的定义，提出证券交易延迟的整体分析框架。对证券交易延迟进行细分，研究构建证券交易系统相关的操作系统、消息中间件、软件开发、性能测试及优化等软件关键技术，讨论可能消除或减少这些延迟的现有方法、技术和手段及其发展趋势。分析表明，采用系统化分析方法对搭建的系统进行不断调整优化是构建低延迟证券交易系统的有效方法。

关键词：低延迟；证券交易；交易延迟；电子交易；消息交换

Research on Key Technologies of Low-latency Securities Trading System

XU Guang-bin[1,2], WU Jian-feng[1], BAI Shuo[1]

(1. Technology Center, Shanghai Stock Exchange, Shanghai 200120, China;
2. School of Computer Science, Fudan University, Shanghai 200443, China)

【Abstract】 Focusing on building the low-latency trading system, this paper gives definition of the securities trading latency and presents the analytical framework and the breakdown of trading latency. The key technologies of operating systems, message middleware, applications, and performance testing and tuning for building a low-latency trading system are studied. In each part of that, existing methods and techniques of eliminating or relieving corresponding latency are discussed and the developing trends of which are provided. It concludes that systematical analysis methodology should be applied and continuously tuning and optimizing should be conducted for building a low-latency trading system.

【Key words】 low-latency; securities trading; trading latency; electronic trading; message exchange

DOI: 10.3969/j.issn.1000-3428.2011.18.010

1 概述

近年来，国际、国内资本市场获得高速发展，带来资本流动性与交易量的急剧增大[1]，使得降低交易系统延迟的需求凸现出来。同时，程序化交易、算法交易、高频交易的大量使用，也造成追求更细粒度交易时间的“竞赛”不断升级。事实上，消息传输带宽、延迟方面的问题已成为证券交易行业近年来发展所面临的最大技术挑战之一。

面对挑战，以纽交所、德交所、加拿大Chi-X交易所、美国Bats交易所为代表的全球主要交易所已经展开了激烈的竞争，纷纷采取措施降低交易延迟，以毫秒甚至微秒来计算证券交易延迟的“低延迟证券交易系统”因此成为研究热点。

本文给出证券交易延迟的分析框架和分类方法，并针对构建低延迟证券交易系统时的软件集成和开发，研究了操作系统、消息中间件、软件开发、性能测试及优化等相关软件关键技术。

2 证券交易系统延迟分析框架

与其他金融服务应用相比，证券交易具有时间集中、数据密集的特点，对交易速度具有很高的要求。以上海证券交易所为例，2009年日均成交股票达到了100多亿股，股票每秒的成交量则达到了数万笔之多，与此同时，每秒钟有数以万计的订单、确认、回报、行情等消息被产生。

如图1所示，证券交易大体上是一个双向往返的过程。广义上的证券交易延迟是指由订单等交易指令从市场参与者系统发出，到交易系统接受、处理，并返回处理结果的时间开销。而狭义上的交易延迟是指交易指令从进入交易系统接入点之后到处理结果返回接入点之间的时间开销。对于证券交易所这种集中式的交易系统来说，客户方系统和网路等外部延迟因素不可控，因此低延迟证券交易系统的构建主要基于后者进行讨论。

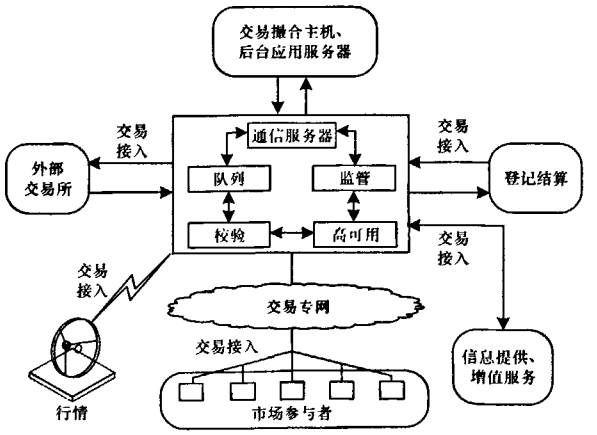

图1 证券交易系统结构

基金项目：中国证监会证券数据压缩传输协议专项研究基金资助项目

作者简介：徐广斌(1976—)，男，博士后，主研方向：证券交易接口技术，金融信息系统；武剑锋，高级工程师、副教授；白 硕，教授、博士生导师

收稿日期：2011-03-02 **E-mail**：gbxu@sse.com.cn

对于证券交易系统，根据经过的处理环节，交易延迟可细分为以下5类：

(1)消息处理延迟：应用消息传输过程中的消息格式转换、消息可用性机制相关的时间开销。

(2)通信处理延迟：主机的协议栈处理开销。

(3)调度延迟：主机提出请求到请求开始被处理的时间开销。

(4)发送/接收延迟：主机向网络发送或接收协议包的开销。

(5)传播延迟：在传播介质上传输的时间开销，主要与传输距离和传输介质相关。

相应地，交易延迟的影响因素涵盖了网络、CPU、存储器、网络接口性能等硬件相关的因素，以及操作系统、中间件(主要为消息中间件)、应用软件等软件相关的因素。如图2所示，这些因素在不同层次上对证券交易延迟造成影响。其中，根据文献[2]，对于高性能、低延迟硬件环境的系统来说，大部分的开销由软件处理所引起。因此，在构建低延迟证券交易系统的时候，尤其需要对这些软件延迟影响因素进行分析，尽量避免或降低延迟影响因素造成的影响，降低交易的延迟。

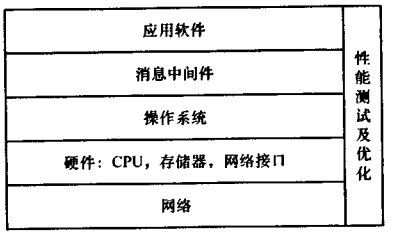

图2 证券交易延迟分析框架

3 低延迟证券交易系统软件关键技术

3.1 操作系统

常见的通用操作系统存在以下不适于低延迟处理的地方：

(1)虽然理论上操作系统时钟最小可以与硬件中断处理的时间相等，但通用操作系统的时间管理一般采用粗粒度的周期性时钟中断。尽管对操作系统本身而言，这避免了频繁进行进程上下文切换，但也使调度的时钟延迟在最坏情况下可能等同于时钟的间隔，从而成为延迟产生的最大来源[3]。例如，Linux缺省时钟间隔粒度为10 ms，因此最坏情况下内核和用户空间应用程序需要使用一个时间间隔来进行调度。也就是说，对于一个休眠的进程，即便唤醒条件已经触发，也许需要10 ms的时间才能被调度执行，这显然难以满足低延迟应用的要求。

(2)通用操作系统中大量存在的非抢占式处理会对低延迟处理造成影响。这是因为，即便操作系统已经使用了细粒度的时钟，而且硬件也及时地产生了一个时钟中断，但如果中断被屏蔽或者内核运行在非抢占式代码区，也会造成内核不能及时中断，应用程序依然得不到及时的调度执行，这样的延迟可能会长达50 ms~100 ms[4]。

(3)线程调度策略也可能造成较大的延迟。即便操作系统使用了细粒度的时钟，并且使用了抢占式方式可以及时中断，但如果优先权不够，还是无法被立刻被调度执行。实时系统领域对该问题进行了较多的研究。其中较好的一类办法是使用按比例的实时调度器[5]，它能按照比例对不同类型的应用线程进行调度，前提是所有线程都是抢占式的，而且使用粒度更细的时钟。

(4)通用操作系统一般都使用虚拟内存，因为这使得并发执行的程序可以在运行程序时，只将运行需要的部分载入内存，从而允许程序空间的总和可以远大于实际可用的RAM内存。对于时分系统来说，虚拟内存不会产生问题。但是对于实时系统，这种页面调度与交换会造成无法容忍的不确定性延迟。

为支持实时、低延迟的应用，一些面向操作系统的实时技术被提出。文献[6]实现的实时操作系统利用MINIX无任何页面调度与交换的特点，使用基于优先权的调度器替换了MINIX操作系统原本基于循环的调度器。该方法对时间粒度的要求不太高的应用比较适合。QNX[7]系统在Unix中使用POSIX.1b实时规范，采用的技术有基于优先权的调度算法、用户内存页面锁定、实时信号、优化IPC和时钟等。Vxworks继承了POSIX.1b中的许多方法，此外，Vxworks中内核和事务进程使用了同一地址空间，这使得事务的切换十分迅速，也避免了系统调用中断。REAL/IX系统[8]兼容POSX.1b，通过实现基于信号量的资源访问替换了传统的休眠/唤醒和中断屏蔽，这有效地降低了中断延迟，并使得向多处理器/多微处理器机器的迁移更加容易。REAL/IX实现的技术包括预分配内存和文件空间、同步和异步I/O支持、用户进程直接处理中断等。Rtlinux[9]系统通过在Linux内核和中断控制器硬件之间加上一层轻量级的虚拟机层来虚拟中断控制器和时钟，从而可以将Linux按照抢占式的方式来运行。其中操作系统内核不能直接控制中断控制器，因为虚拟机层使用宏替换了所有开/关中断和中断返回指令，所有的硬件中断均被虚拟机层捕获，然后根据中断状态判断是否需要处理该中断。由于内核比实时应用的运行优先权低，就不会对不被虚拟机捕获的实时中断造成影响。为支持实时和低延迟的应用，Redhat公司也于2007年底推出了面向实时和高性能消息交换的Redhat MRG[10]。MRG采用了更细粒度的时钟，以信号量代替锁，减少了压缩非抢占式操作的代码，用线程处理中断、读-拷贝修改等各种技术来支持低延迟处理。根据Redhat在2009年峰会上公布的数据，配置为24 GB内存、2.93 GHz Xeon四核处理器、Infiniband 4XQDR接口的MRG单机上每秒消息通信量已突破百万级。为满足实时、低延迟需求，微软公司也推出了支持实时的Windows CE操作系统。其他的实时操作系统还包括Ecospro、RTEMS、EROS、Freertos、Ecos等。

在多核处理器成为主流之后，利用多核系统并行运行多线程以最大化计算机的处理效率成为研究的热点。在这种系统中，线程被尽可能地分派到不同的核心上执行。同时，对于不希望被别的线程打扰运行的线程，或一些访问特定资源(如内存、I/O)的线程、以及一些对无需上锁的运行时资源进行管理的线程，可以利用操作系统线程关联的机制，将线程绑定到特定的处理器，加快程序的运行速度。

3.2 消息中间件

消息中间件(MOM)集运行系统、管理工具集和开发系统于一身，既为上层应用系统提供了可靠、高效、易扩展的数据通信方式，又为网络系统提供了实时管理和监控的工具，具有高效、可靠、功能强、跨平台、跨网络等优点，可用于交易系统内部组件、子系统、以及外部系统之间的消息通信。

MOM常被用来屏蔽掉各种平台及协议之间的特性，实现分布式计算环境中的跨平台数据传输。MOM的通信模式

主要包括点对点模式、发布/订阅模式和消息队列模式。消息队列模式通过消息队列进行通信，是程序之间一种非直接的通信模式，并不要求与对方程序建立逻辑链接。在消息队列模式中，消息队列管理器负责消息队列的可持续化存储、负载均衡、消息确认、优先级、条件触发、多路复用等功能。点对点模式是程序之间的一种同步通信方式。在双方程序之间需要建立逻辑链接，应用请求通过消息的形式直接从一个程序发送到另一个程序。点对点模式并不适合松耦合、时间独立的应用。发布/订阅模式将应用程序分为进行信息发布的应用程序，以及接受某个特定主题信息的应用程序。发布消息的应用程序只需要简单地将消息以主题方式发送出去，由消息代理来负责将消息传递给所有订购该主题的订阅消息的应用程序。因此，发布/订阅模式是一种异步、松耦合的通信模式。其他的通信模式还包括：请求/应答模式，扇出模式等。MOM能够在客户和服务器程序之间提供同步和异步的连接、订阅与分发，并且可以对消息进行传送或者存储转发，还可以跟踪事务，并且通过将事务存储到磁盘上实现网络故障时系统的恢复。

大多数消息中间件都为为编程人员提供了编程接口，如AMQP的一些中间件则提供了基于公开的协议消息格式的调用方式。主要的中间件产品包括 IBM MQseries、Tibco Rendezvous、基于 JMS 的 JMQ、微软的 MSMQ、29West、RMDS、BEA 的 Messageq，以及 Tonglink/Q、开源的 AMQP、A2E-MQ、Ezbridge 等。在交易接入系统实际应用中，可以根据需要对的消息中间件技术进行灵活地借鉴、整合，以满足低延迟应用的要求。

3.3 应用软件延迟

应用层延迟与具体的应用以及运行时状态紧密相关，降低这部分处理产生的延迟除了要靠提高硬件计算能力外，还需要提高软件的处理效率，并根据应用的特性来调整系统的性能，这可包括：

(1)利用并行计算加快软件处理速度。应用层处理的开销主要是由消息处理所包含的错误检测和恢复、数据格式转换、消息路由、消息流控、消息的发送和接收所引起的。如前所述，多处理器/多核计算机目前已经成为主流的技术，因此可以利用现有并行计算的相关技术，提高并行化程度来充分利用多处理器/多核的计算能力。Openmp 是一个可用于并行系统和单系统的并行化编程模型，包括一套编译指示语句和一个支持的函数库。其中，编译指示描述了程序应该以何种方式并行执行，函数库则是为并行化提供支持。Openmp 移植方便，支持 C、C++、Fortran 等标准语言，可在不同硬件平台和操作系统上执行。MPI 是消息传递函数库的标准规范，具有上百个函数调用接口库，可支持多系统环境下的并行进程之间的通信和同步，它独立于编程语言，可直接在 Fortran 和 C 语言中可以直接对这些函数进行调用。此外，MPI 也是一种跨硬件平台和操作系统的标准，使用 MPI 进行消息传递的 C 或 Fortran 并行程序可不加改变地运行在 IBM PC、MS Windows、Unix 工作站、以及各种并行机上。Pthreads 是 IEEE Std 1003.1c-1995 定义的一个用于创建和操作线程的 API 标准。通过 Pthreads 的多线程技术可实现程序在多处理器上的并行化。与标准的 Fork()多进程技术相比，线程带来的开销很小，内核无需单独复制进程的内存空间或文件描述符等，这就节省了大量的处理器时间。线程共享相同的内存空间，支持内存共享无需使用繁琐的 IPC 和其他复杂的通信机制。Pthreads 也是一种跨平台易移植的并行化技术，可用于大部分的类 Unix 系统，如 Freebsd、Netbsd、GNU/Linux、Mac OS X and Solaris，也支持 Microsoft Windows。

(2)利用专用的软件开发或运行平台加快应用处理速度。对于低延迟有要求的计算服务器或者通信服务器等应用，由于对运行速度的要求高、对计算机底层技术的操作性强，一般采用 C 及 C++等中高级语言对软件编制，而不使用 Java、.NET 等基于虚拟机的高级语言。为了支持实时及低延迟的应用。基于 Java 的实时性的 RTSJ 规范被提出来。RTSJ 对 Java 虚拟机和 Java 库进行了扩展，包括线程同步、优先级扩展、用户定义调度算法、异步事件处理、物理内存直接访问、内存管理扩展、时钟粒度控制等。在 RTSJ 中，为了处理垃圾回收导致无可预测的延迟，实时线程可以使用作用域内存，不会被垃圾回收处理。基于 RTSJ、Java RTS、Jrate 等 Java 实时平台被开发出来。文献[11]对 Java RTS 和普通 Java 平台进行了测试，结果显示前者在平均响应时间、最大响应时间上都要优于后者。基于微软.Net 框架的实时开发和应用也正为研究所关注[12]。

(3)借鉴 QOS 的方法保证关键应用的处理速度。借鉴传统 QOS 的方法，可根据对延迟要求的不同来区分处理业务数据，比如哪些可以不传或者少传、哪些是必须在线传或离线传、哪些处理逻辑是可以通过放到硬件处理来加速等。对于消息流的传输也可以通过业务流控制和拥塞控制措施，来增强对延迟的可控性，这不仅可保证对延迟具有较高要求的数据能够根据特定的策略优先使用计算和网络资源，还可防止极端情况下海量突发消息流对接入系统和整个交易系统的冲击。

(4)利用虚拟机技术可灵活利用计算设备处理能力。近年来，以 Disco、Vmware、XEN[13]、KVM、Qemu、Virtual PC、UML 等为代表的虚拟机技术成为学术界和工业界所共同关注的研究热点。利用虚拟机技术不仅可以在同一硬件上虚拟出多个完整独立、相互隔离的计算机运行环境，同时可以按需地对实际计算机资源进行动态调整。在构建低延迟系统时，可以利用虚拟技术，根据不同应用层业务特点及需求的变动进行灵活的计算机软、硬件资源调配，消除延迟瓶颈、加快处理速度。

3.4 性能测试与优化

3.4.1 性能测试

性能测试一般是借助各种软件或硬件工具对计算机的特定软硬件组件进行速度或效率的测量，是以各种指标对系统性能进行评价的一个重要环节。对低延迟交易系统，除可以对满足实际应用需求指标的情况进行测量，也需要结合利用多种标准或常用的测试工具来测量，以便在多个维度上进行评价和比较。常用的计算机测试工具如表 1 所示。

表 1 常用的计算机测试工具

分类	典型软件/工具
网络及 TCP	Netperf, NETIO, Webbench, Openwebload, Jmeter, Iperf
文件系统及 I/O 子系统	Iozone, Iometer, Bonnie++
处理器及内存	LAPACK, Specfp, Ubench: Super PI
综合	Pcmark, Lmbench, SPEC 系列
多处理器/多核计算	EEMBC
压力测试	Prime95, Stress
能耗	DCIE

3.4.2 动态调试及性能调整

传统的性能测试基本上是一种静态、粗粒度的方法，难以对相关资源在应用运行过程中的情况变化进行动态的揭示。动态跟踪可以在应用运行时自动进行实时监控、分析和调节，极大地方便了错误排查和性能瓶颈的检测。动态跟踪一般通过在程序代码中嵌入观测点，可以提供比传统测试更小的监控粒度。根据监控对象层次可以将动态跟踪的对象按照应用层、标准库、操作系统、硬件几大类进行划分，具体则包括程序运行时的 TCP/IP 栈、对处理器、内存、BIOS、文件系统、网络、I/O、算法、负载均衡等。动态跟踪是近年来是系统领域的研究热点，具有代表性的动态跟踪工具有类 Unix 平台上面向并行计算和实时应用 Dtrace，用于 Linux 平台的 Dprobes、Lttng，用于 AIX 平台的 Probevue，面向 Openmp 应用的 Omptrace 等。用于 Windows 平台的动态调试方法 Ntrace 也在最近被提出。

此外，基于虚拟机的技术可全真地对运行过的应用进行重放，从而进行进一步的调试和性能调优。虚拟机技术可以实现指令粒度的监控和捕获，从而实现与运行时完全一样的情景的全过程重放，每次重放也完全一致。对于传统测试以及动态追踪技术难以捕获、重现及描述的非确定性事件，可以通过虚拟机技术以完全相同的方式重现。基于虚拟机重放的动态调试方法有 Retrace、Revirt、SMP-Revirt 等。

3.4.3 业务数据流分析及仿真

数据挖掘和机器学习技术可用于分析、识别业务数据流的模式，可作为不同种类业务数据流传输模式和服务质量的基础和依据，优化整体处理延迟。此外，结合测试及调试技术，可挖掘时间相关的业务数据流与时间或处理开销“热点”之间的关系，使得延迟优化更加动态、自适应。

利用仿真可进一步消除潜在的延迟瓶颈。如果只是对历史数据进行分析，难以发现因为未来交易产生业务数据流影响下，会造成的性能“热点”与延迟瓶颈。借助虚拟市场和市场仿真技术可以实现“机器人报单”，实现对未来市场行为的预测和仿真。传统的基于数学方法的市场仿真技术主要通过学习历史数据抽象出数学模型，并借此来推测未来的市场行为。但是，证券交易是个复杂系统，由异构、交互、投资策略出不穷的投资者来决定。这造成市场不断变化，单一规律或模型的可重现率不高。针对传统数学方法进行市场仿真的不足，基于 Agent、多 Agent 等的动态市场仿真技术[14-15]被提出。与基于数学模型的方法相比，这类方法更加强调投资者的动态和交互性，因此可以产生更加贴近真实市场行为的业务数据。各种市场仿真技术可作为现有的性能测试和优化方法的有益补充，用于进一步发现和消除潜在的延迟瓶颈。

4 结束语

数十年来，计算机硬件能力大致保持了每 18 个月就翻一番的规律，即著名的摩尔定律，然而软件性能的提高却远滞后于这个速度。因此，在构件低延迟证券交易系统的时候，除硬件系统的搭建需要满足低延迟的需求外，软件系统在设计、实现和集成的时候，更需要考虑避免或减少引入延迟影响因素。

本文重点讨论了延迟原因、可能消除或减少延迟的现有技术、方法和发展方向。然而，交易接入系统是个多因素相互作用的复杂系统，各部分对交易接入延迟的影响基本上符合 Amdahl 定律——单独对系统一个部分的改进并不一定能带来整体性能的提升和延迟的下降。因此，打造低延迟交易系统要把握均衡和谐的原则，在抓住关键因素的同时、要从系统整体作考虑，细致分析各种因素的相互影响以及与周边系统的相互联系，结合实际不断调整优化。

参考文献

[1] Agarwal V, Bader D A, et al. Faster FAST: Multicore Acceleration of Streaming Financial Data[J]. Computer Science: Research and Development, 2009, 23(3): 249-257.

[2] Karamcheti V, Chien A A. Software Overhead in Messaging Layers: Where Does the Time Go[C]//Proc. of International Conference on Architectural Support for Programming Languages and Operating Systems. San Jose, USA: [s. n.], 1994.

[3] Abeni L, Goel A, Krasic C, et al. A Measurement-based Analysis of the Real-time Performance of the Linux Kernel[C]//Proc. of the 8th IEEE Real-time and Embedded Technology and Applications Symposium. San Jose, USA: [s. n.], 2002.

[4] Goel A. Operating System Support for Low-latency Streaming[D]. Portland, USA: School of Science and Engineering, Oregon Health and Science University, 2003.

[5] Barabanov M, Yodaiken V. Real-time Linux[J]. Linux Journal, 1997, 34(2): 19-23.

[6] Wainer G A. Implementing Real-time Services in MINIX[J]. Operating Systems Review, 1995, 29(3): 75-84.

[7] Hildebrand D. An Architectural Overview of QNX[C]//Proc. of the USENIX Workshop on Micro-kernels and Other Kernel Architectures. Seattle, USA: [s. n.], 1992: 113-126.

[8] Furht B, Parker J, Grostick D. Performance of REAL/IXTM-fully Preemptive Real Time Unix[J]. ACM SIGOPS Operating Systems Review, 1989, 23(4): 45-52.

[9] Yodaiken V. The RTLinux Manifesto[C]//Proc. of the 5th Linux Conference. Raleigh, USA: [s. n.], 1999.

[10] Redhat, Inc.. Redhat MRG[EB/OL]. (2010-04-08). http://www.redhat.com/mrg.

[11] Doherty B P. A Real-time Benchmark for Java[C]//Proc. of the 5th International Workshop on Java Technologies for Real-time and Embedded Systems. Vienna, Austria: [s. n.], 2007.

[12] Zerzelidis A, Wellings A J. Requirements for a Real-time .NET Framework[J]. ACM SIGPLAN Notices, 2005, 40(2): 41-50.

[13] 胡冷非，李小勇. 基于Xen的I/O准虚拟化驱动研究[J]. 计算机工程, 2009, 35(23): 258-262.

[14] Viamonte M J, Ramos P. Simulating the Behaviour of Electronic MarketPlaces with an Agent-based Approach[C]//Proc. of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence. Beijing, China: [s. n.], 2004.

[15] Praca I, Viamonte M J. Agent-based Simulation of Electronic Marketplaces with Decision Support[C]//Proc. of the 2008 ACM Symposium on Applied Computing. Fortaleza, Brazil: [s. n.], 2008.

编辑 任吉慧

低延迟证券交易系统关键技术研究

作者： 徐广斌， 武剑锋， 白硕， XU Guang-bin， WU Jian-feng， BAI Shuo

作者单位： 徐广斌,XU Guang-bin(上海证券交易所技术中心,上海200120;复旦大学计算机科学技术学院,上海200443)
， 武剑锋,白硕,WU Jian-feng,BAI Shuo(上海证券交易所技术中心,上海,200120)

刊名： 计算机工程 ISTIC PKU

英文刊名： Computer Engineering

年，卷(期)： 2011,37(18)