自治组件架构在存储业务仿真测试中的应用

张 毅 文俊浩 陈 义
(重庆大学软件学院 重庆 400044)

摘 要 存储产品的吞吐量、响应时间、支持并发数等常规性能指标及其用于具体业务领域的性能表现体现了其核心竞争力。因此,进行性能测试是存储产品研发和采购活动的重要组成部分,高效和高置信度的存储性能测试软件是存储厂商和客户的迫切需求。然而,目前业界常用的测试存储性能的软件存在一些缺陷,如应用场景有限、测试结果失真等。针对上述不足和当前的需求,通过采用自治组件架构(ACA)的思想,利用其标准化、松耦合、可重用性、可扩展性等特征,提出了一种基于业务场景仿真方式的存储性能测试系统的解决方案,分析了通用性能测试软件的技术原理与系统架构,提出了自治组件架构的业务仿真性能测试软件的系统模型,阐述了基础设施与组件通信流程的设计,给出了系统主要业务模块的实现原理。

关键词 自治组件架构,业务仿真,存储性能测试
中图法分类号 TP311.5 **文献标识码** A

Application of Autonomous Component Architecture in Storage Business Simulation Test

ZHANG Yi WEN Jun-hao CHEN Yi
(School of Software Engineering, Chongqing University, Chongqing 400044, China)

Abstract The general performance indicators of the storage product such as throughput, response time, concurrent number of I/O, and its performance under the specific applications reflect the core competitiveness. Therefore, performance testing is an important part of the storage products' development and purchase. The storage performance testing software with efficiency and high degree of confidence is the urgent need for both of the manufacturer and the customer. However, so far the usual software in the industry has some deficiencies, namely, limited scenarios, distortion of testing result, etc. Aiming at these shortcomings and current requirements, the thesis making use of autonomous component architecture (ACA) with its standardized, loosely coupled, reusable and scalable features, proposed and implemented a feasible solution of a scenario simulation storage performance testing software. Then, the technical principle of general performance test tools and the architecture of system were analyzed. The system model of the business simulation performance test tools was presented for autonomous component architecture. The infrastructure and the component communication were designed. The implementation principle of the main modules was proposed.

Keywords Autonomous component architecture (ACA), Business simulation, Storage performance testing

1 引言

存储产品的吞吐量、响应时间、支持并发数等常规性能指标及其用于具体业务领域的性能表现体现了其核心竞争力。进行性能测试是存储产品研发和采购活动的重要组成部分。一般来说,存储产品的性能测试工作都是通过自动化测试来实现的。因此,高效和高置信度的存储性能测试软件是存储厂商和客户的迫切需求[1]。

目前,广泛使用的存储性能的测试方法是业务场景仿真测试,即使用自动化测试软件模拟各种类型的业务场景来测试存储,如 SPC(Storage Performance Council)基准程序、SPEC (Standard Performance Evaluation Corporation)基准程序等。然而,业界常用的测试存储性能软件均存在一些缺陷,如测试代价高、测试结果失真、应用场景有限、用户体验差等。

针对存储性能测试软件的不足和当前的需求,通过采用自治组件架构(ACA)的思想,利用其标准化、松耦合、可重用性、可扩展性等特征,提出了一种基于业务场景仿真方式的存储性能测试系统的解决方案[2]。

2 应用研究的相关理论

2.1 自治组件架构的概念

自治组件架构(ACA, Autonomous Component Architecture)的设计思想是,软件系统由若干个自治组件(component)组成,每个组件具有若干个端口(port)。组件之间的通

到稿日期:2011-10-20 返修日期:2011-12-30 本文受国家自然科学基金(70472015),重庆市自然科学基金(CSTC2008AB3014)资助。
张 毅(1962—),男,博士,副教授,主要研究方向为软件测试技术和嵌入式系统设计与开发,E-mail:cquzhangyi@163.com;文俊浩(1969—),男,博士,教授,博士生导师,主要研究方向为服务计算与面向服务的软件工程等;陈 义(1984—),男,硕士生,主要研究方向为软件设计与开发。

信方式由协议(contract)来规定,协议既可以绑定到一个端口(port),也可以绑定到一个组件(component)[1]。端口与端口之间是通过导线(wire)来连接的,也就是组件之间传送的信息(message)按照协议规定,通过导线传递到对端组件的端口。

自治组件架构类似于工业中集成电路的设计架构,就像集成电路是由若干个 IC 芯片(即集成芯片)装配到集成电路板而组成的[3]。每个 IC 芯片相对于外界完全是黑盒的,它由指导手册中的功能规范及相关信号输入/输出模式来定义,输入信号的变化会触发它按照规范执行特定功能并作相应的输出。事实上,IC 芯片仅仅是通过针脚与其他芯片、模块或者系统来交互。可以说,每个 IC 芯片均是独立的,可以针对单个 IC 芯片进行设计、开发和测试。自治组件架构是模仿集成电路的设计和制造模型的,用相似的概念定义了软件组件的设计和集成方式。软件系统的组件类似于 IC 芯片的概念,而组件的功能与对外接口(输入/输出)则由类似指导规范概念的"协议"来规定。不同的组件被集成到同一系统下,由一个类似于线路板的运行或者模拟器来接管组件之间的通信。与传统的软件系统架构相比,自治组件架构的组件之间是完全透明的,它们仅对协议负责。

2.2 自治组件架构中组件、协议和端口的定义

自治组件是自治组件架构最基本的组成部分,每个组件包括若干个对外通信的接口,称之为端口。某个端口所驻留的组件被称为该端口的宿主组件。协议规定了发起者组件和响应者组件如何实现某个特定的功能,并规定了二者通信的格式等。组件和端口均使用名称来标识其唯一性,它们的命名方式类似于现代操作系统中的文件系统的层次名称规范[4]。

组件之间的通信过程,首先保证组件之间的通信是互连的。自治组件架构中,端口与端口之间是通过导线(wire)来连接的,两个端口连接便代表它们各自的宿主组件也连接起来了。其中,一个组件将数据发送至其包含的某个端口,然后由该端口将数据转发至与其互连的端口。当数据到达目标端口时,该目标端口的宿主组件将立即启动一个线程来处理该数据,并可以通过与其相应协议绑定的端口输出处理结果。输入和输出端口分别通过各自的导线连接,即与端口绑定的导线分为输出导线和输入导线两种。比如,当某个端口发送一个数据时,数据将被发送到该端口的输出导线,再被发送到将该导线设置为自己的输入导线的所有端口上,才交由这些端口的宿主组件处理。由此可见,该过程与集成线路中通过物理导线传输数据的形式相似。端口互连分为一对一、一对多以及多对多 3 种情况。图 1 给出了端口的连接模式。这些交互方式基本上满足了不同领域中所需要模拟的业务组件之间通信的所有构成形式[5]。不过,自治组件架构不支持端口自身形成闭环,所以当端口的输出导线与输入导线为同一导线时,数据输出并不会被自身接收。同时,端口之间的互连分为单工模式和双工模式。如果 A 端口的输出导线和 B 端口的输入导线互连,仅允许 A 向 B 传输数据,则称这两个端口连接模式为单工模式;双工模式是指 A 端口的输出导线与 B 端口的输入导线互连时,A 端口的输入导线也与 B 端口的输出导线互连,此时 A 和 B 能够互传数据。

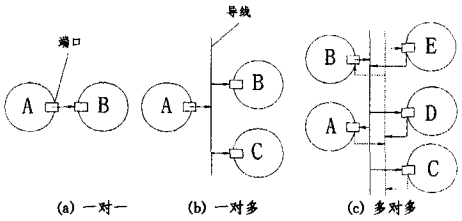

图 1 端口的连接模式

协议并不直接对通信双方的组件本身进行规定,而是定义了组件间发送或接收的数据的因果关系,即消息的先后关系、格式、状态等。协议也可以分为两种类型,即端口协议和组件协议。端口协议指绑定到特定组件端口的协议,而一个组件协议则描述了组件如何针对传送至它的所有端口的数据进行响应。

2.3 自治组件的优势和缺陷

① 协议设计时,通过绑定和组件集成方式,可以使组件独立于软件系统的其他组件/模块进行开发和测试,并且,软件系统可以经由组件的增量式集成来构建。这既降低了软件模块间的耦合性,又提高了软件组件的复用性[6]。

② 一般模式下,当数据到达某个组件的某个端口时,将采用独立的线程处理数据,这样的处理方式,可不用再关心不同数据同时到达某组件的问题。反之,如果使用单线程,就需要编写反复跳转来判断新数据的到达并编写相应的处理程序,这比较繁琐而且易出错,同时执行效率较低。

③ 业务逻辑与执行引擎分离。自治组件架构中的运行时(Runtime),亦即组件通信和任务执行的执行引擎。其作用类似于集成电路中的电路板,它负责注册组件的消息处理线程的创建、管理等任务。自治组件程序员可以不关注执行引擎的具体实现,只需要专注于组件业务逻辑的实现。而执行引擎则可由第三方提供或者由特定人员进行开发、维护。执行引擎一经产品化,便可以在其他自治组件架构的系统中被复用[7]。

综上所述,自治组件架构适合于广义流程清晰、粒度化,后续面临策略或算法的增加、修改、替换,需要进行并行开发的软件系统。比如它适合于网络仿真、软件性能测试等软件系统。采用自治组件架构,实现了组件之间的解耦,使设计人员可以更抽象地实施系统设计;使组件可以并行地进行开发、调试和测试;使系统集成更加简单;使系统能够更灵活地应对需求变更。然而,自治组件架构也存在一些缺点,如程序员不得不实时重视多线程带来的资源同步问题,以及执行引擎是否健壮和高效直接决定了整个系统的优劣。

3 存储业务仿真测试系统组件模型

3.1 系统组件模型

一个通用的存储性能仿真测试场景包含以下基本元素:虚拟用户、虚拟目标资源及虚拟操作。一个存储性能测试用例是众多基本元素的组合。需要对这些基本元素进行管理,同时实现对基本元素的调度,这就给出了系统模块:用例生成

器模块、虚拟用户模块、虚拟资源模块、虚拟操作模块、监测器模块、控制模块，以及执行引擎模块等。与业务流程相关的模块设计成自治组件(控制模块除外)，包括用例生成器组件、虚拟用户管理器组件、虚拟资源代理组件、虚拟操作代理组件、监测器组件。它们的作用分别是测试用例解析和生成、虚拟用户管理和任务调度、虚拟资源管理、虚拟操作管理与负载调度。而执行引擎虽然是基础设施的一部分，与具体业务无关，但是它也需要根据具体的应用来定制，以达到最大的运行效率[8]。图 2 所示为系统组件模型图。

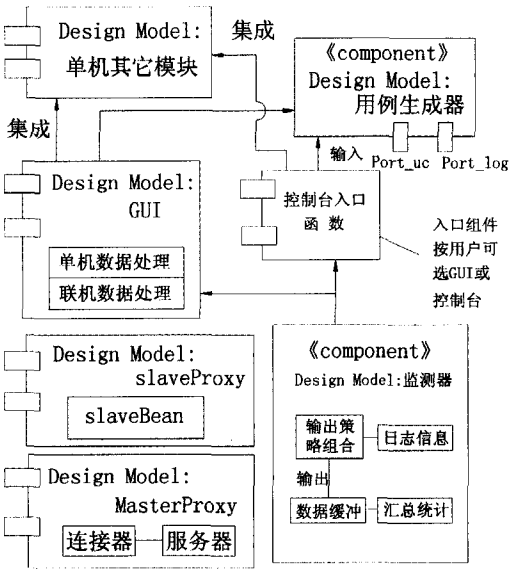

图 2　系统部分组件模型

3.2　系统组件交互模型

系统中与业务逻辑相关的模块包括管理型自治组件：用例生成器组件、虚拟用户管理组件、资源代理组件、操作代理组件和监测器组件。这些自治组件在设计时绑定相应的端口与协议，然后在系统集成时形成组件之间的绑定。在设计时，需要设计不同组件之间的关联协议及其相应的职责，要求统筹地设计不同职责的自治组件之间的交互模型[9]。

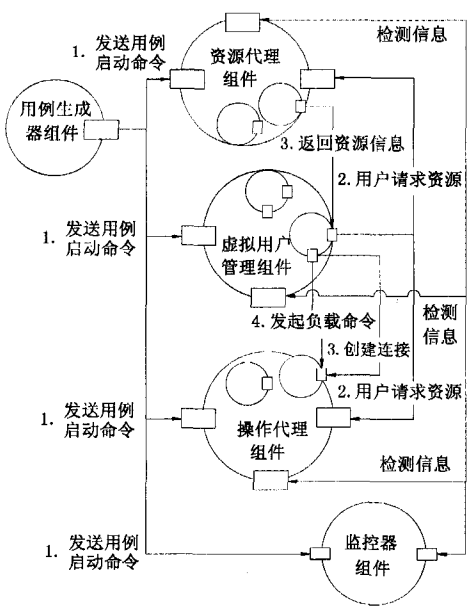

图 3　测试用例执行组件交互模型

图 3 给出了测试用例执行时，系统各个自治组件之间的交互关系和步骤。组件或端口之间的连接均依赖于相关协议，当用户控制端(控制台或用户界面)启动测试之后，将会产生一系列的组件间交互。下面介绍主要交互模型和过程。

① 用例生成器组件绑定预处理协议，并通过该协议与其他 4 大组件进行互连，即运行时关联。首先，用例生成器组件将解析配置文件并生成相应的测试用例对象。该用例对象中不仅包括了用例的操作参数，还包括成功执行该用例所需要的最小资源实例个数、操作实例个数等前置信息，该对象的内容就是预处理协议规范的消息内容。然后，用例生成器组件将主动把用例对象发送给其它 4 大组件，通知它们进行相应的预处理操作。此时，这 4 大组件将并行进行用例执行前的预处理操作：资源代理组件进行目标资源/设备准备等工作；虚拟用户组件进行用户实例准备、任务容器初始化、调度策略实例化等工作；监测器组件进行监测数据容器初始化、时钟初始化等工作；操作代理组件进行操作实例准备等工作。

② 虚拟用户组件绑定资源路由协议、操作路由协议等，并通过这些协议与资源代理组件、操作代理组件、操作组件等进行互连。虚拟用户组件实例由虚拟用户管理组件创建和管理。当虚拟用户管理组件接收预处理信息，并使其虚拟用户组件实例对象个数达到最低要求时，其状态变为“准备就绪”状态。然后，用户管理组件将命令用户组件实例查找对应的目标资源以及操作实例。用户组件便按照资源路由协议、操作路由协议规范的消息格式，分别发送消息至与之连接的资源代理组件和操作代理组件，并分别从这些管理组件的组件对象池中查找相应的对象，虚拟用户组件实例与其对应的目标资源组件实例和目标操作组件实例建立连接。

③ 虚拟用户组件和与之对应的资源组件实例及操作资源组件实例连接之后，该资源组件按照资源控制协议的规范发送相应资源信息给虚拟用户，虚拟用户准备完毕，进入请求调度队列，等待负载启动命令。

④ 虚拟用户组件按照操作控制协议的规范，向与之连接的操作组件发送相应的操作参数并发起负载启动命令。每个操作组件都封装了用例中定义的特定业务类型的特定用户操作的 I/O 负载策略，当其接收到用户组件的消息之后，便按照给定的参数下发相应的读写 I/O 到存储，并不断地执行 fork 操作来重复执行读写 I/O 操作，直到完成用例规定的时长或任务；最后再发送消息通知虚拟用户组件当前的执行状态。

此外，在用例执行过程中，监测器组件始终是与其他相关组件保持连接的，它周期性地向其他组件发出收集监测指标数据的命令，并针对接收的数据执行组合、计算、统计与输出等任务。同时，提出了子用例的概念。一个测试用例中，可能包括若干个起始时间和执行时间不相同的任务。将具有相同运行参数的任务归类为一个测试子用例，提出子用例的概念是为了方便用户配置模拟真实场景的测试用例，同一测试用例的不同子用例的任务之间仅是任务的操作类型、起始时间和执行时长不同，它们的调度策略和调度流程是一致的。

⑤ 每个操作组件执行完任务后，便发送操作控制协议规范的消息给虚拟用户组件，表明自己操作结束。虚拟用户接

收到此信息后便，修改自身的状态并通知用户容器。因此，可由用户管理组件中的活跃状态的用户数来判断用例的执行进程。当前用例执行结束时，用户管理组件向监测器组件发送停止监测消息，同时向用例生成器组件发送用例执行结束消息。然后，用例生成器组件判定停止测试或者发送下一个测试用例给其他组件，即开始新用例的执行流程。

4 目标仿真业务的构建与实现

4.1 目标仿真业务的构建

自治组件架构下，目标仿真业务的各种实体或元素均被定义为组件。如果目标仿真业务为 VOD(视屏点播，Video On Demand)业务，那么需要构建 PLAY 操作、DRAG 操作、PAUSE 操作等组件，这些组件分别实现了相应操作的 I/O 模型，同时需要添加解析此类业务的操作字典的测试用例解析组件。

相应地，如果需要支持电子邮件业务仿真测试，则仅仅添加 CREATE、WRITE、DELETE 等操作组件，以及要求测试用例解析和生成组件支持解析该类型业务的操作字典规定的文本[10]。

目标仿真业务扩展时，无需修改系统框架以及其它仿真业务的组件和协议，系统支持各种目标仿真业务的扩展，如 NVS、数据库业务、视频监控业务等，其可以发展为应用于各种类型业务存储产品的性能测试平台。

4.2 基于自治组件架构的存储测试系统实现

自治组件架构只是一个软件系统架构的方法论，它并没有定义具体实施细节，采用自治组件架构构建存储测试系统必须开发或选择第三方来实现自治组件架构规范的系统框架。

下面以监测器模块的实现过程为例，说明基于自治组件架构的存储测试系统的实现方法。监测器模块负责实施实时性能监测，它实时获取当前系统各个组件与业务相关的性能数据，并周期性对其进行汇总和输出。监测器模块主要由 Monitor 组件构成。但是，要实现系统实时监控功能，需要各种类型的组件都具有信息统计功能，由 Monitor 组件来实施汇总和输出等。

4.2.1 性能指标类别和设计要求

将需要监测的信息项目统称为监测指标，把具有相近意义的指标归纳为一个类别，统称为统计类别。需要监测或计算的统计类别包括用户池信息、操作池信息、资源池信息、通用性能信息和用户评价信息等[11]。

不同业务类型的测试用例，所要求模拟的操作等是不同的，其所需监测的统计类别是不同的，而且这些统计类别中包括的指标也是不同的。例如，只有 VOD 或 IPTV 业务要求统计用户评价信息；如果某 VOD 测试用例要求模拟用户 PLAY(播放)、DRAG(拖动)、REPLAY(重播)操作，则该用例的操作数信息则仅需要包含这 3 个监测指标，而不需包括 FF(快进)、FB(快退)等。同时，不同的测试用例的执行时长不同，用户的关注点也会不同，如长时间的测试(如 7×24 小时)可能需要将过程数据和汇总数据写入数据库或文件中，以便用例结束之后分析和查看，而无需在屏幕上进行实时输出。短时长的测试则可能正好相反。

4.2.2 实现

为了满足不同的业务类型的要求以及提高执行效率，开发实现了统计类别由人工设置和自动组装相结合的功能，数据输出形式则仅由用户设置。为了避免统计任务对负载操作任务的影响，采用了两级统计策略[12]。

一级统计是指各个管理型自治组件收集和累计不同统计类别的指标数据，并周期性地统计这些数据，将其包装并发送给二级统计模块，然后由二级统计类进行实时输出和缓存。当缓存的信息组个数达到用户配置的个数时(以下统称为缓冲数，默认为 20 次，即指 20 次周期为一轮)，由其进一步对这组数据进行汇总计算和输出。同时，这个汇总结果与上一次的汇总结果再次汇总便得出用例启动至当下的各个指标的数值。这样，该测试用例执行结束时便得出整个用例的各项综合指标。

① 一级统计的实现

用户关心的不同统计类别在系统的设计之初已经被隔断在不同的模块之中，即某个统计类别中涉及的所有监测指标均只会在唯一模块中被监测，而同一模块中可能监测多个统计类型。因此，为每一个系统模块实现一个单态的监测类，该类的监测函数仅允许该模块调用，负责统计周期内涉及该模块的监测指标[13]。

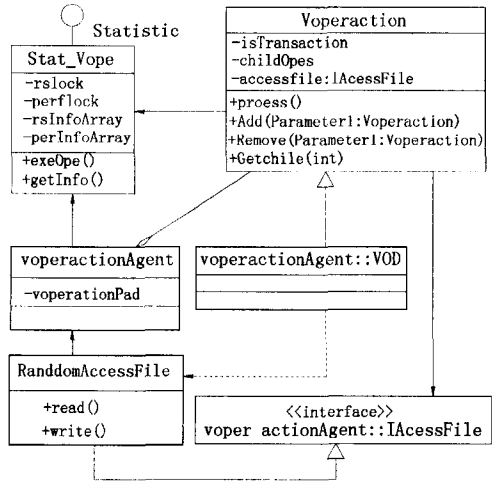

图 4 监测器模块的一级统计类图

监测器模块需要监测的类别包括操作信息和通用性能信息，某些业务场景下还需要统计用户评价信息。如图 4 所示为监测器模块一级统计类图，类 Stat_Vope 是负责模块统计的单态类，Voperation 的继承类业务操作将通过调用 IAccessFile 接口实现类的 create、read、write 等基本的 I/O 动作的组合来实现。该业务操作执行一次 read 动作就会调用 Stat_Vope 中的同步函数 exeOpe()将 IOps 指标递增 1，如果 IAccessFile 的实现类的 I/O 模式为非直接阻塞 I/O，那么还可以计算出该 I/O 的响应时间，然后求出当前最大响应时间、最小响应时间以及平均响应时间等。当 VoperationAgent 组

件接收到 Monitor 组件的周期消息时，它首先统计当前周期的监测信息，如计算最大值、最小值、均值等，然后再将数据包装并发送至二级统计组件 Monitor，同时开始下一个统计周期，故称为一级统计。

② 时钟调度和监测策略装载

图 5 所示为监测器模块的时钟调度和监测策略类图，Catalog 表示统计类别的抽象类，其子类表示各种统计类别获取该类别周期内监测指标等信息的策略；CatalogCol 代表当前用例统计类别的策略集合；StaCtrl 类的作用是根据测试用例和用户配置初始化 CatalogCol，以及用于时钟调度的计时器对象(timer)和触发器对象(trigger)，并负责周期性地执行 CatalogCol 实例中包含的 Catalog 策略，即用于生成向 Monitor 组件连接的组件发送的汇总命令[14]。

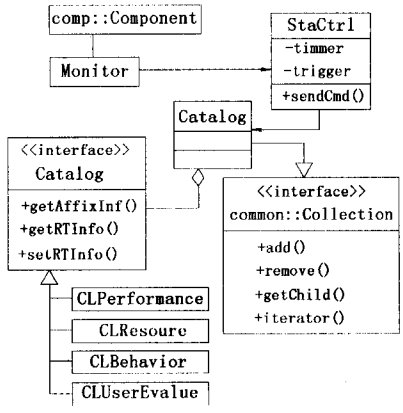

图 5 监测器模块的时钟调度和监测策略类图

③ 二级统计的实现

二级统计的功能主要在 Monitor 类中实现。Monitor 主要负责数据缓存、汇总/统计以及输出，这里的输出包括 3 层输出：一是实时输出，即根据用户设置的周期将数据输出到控制台或绘制曲线图到用户界面；二是延迟输出，即当用户设置的二级缓存被填满时，将整块数据输出到文档或数据库；三是测试用例执行结束之后统计和分析信息的输出[15]。用户可以根据实际情况设置输出目的和输出类型。这里的实现方式是策略模式和组合模式结合的形式，只是用户可以放弃实时输出或者延迟输出，同时用例结束后汇总信息的输出方式是前两层输出策略的并集。为了缓存数据，实现了一个 DataSet 的数据结构，DataSet 由多个 DataTable 组成，支持多线程对 DataSet 中的 DataCell(单元)的读写。存储数据时，每种监测指标对应一个 DataRow，每种统计类别对应一个 DataTable，当前用例的所有统计类别组成一个 DataSet。数据表中的行的数据表示每个周期的监测值。当 DataSet 的数据累计至缓冲数时，便进行一次汇总和输出，然后清空 DataSet 并接收下个缓冲周期的数据。

结束语 针对目前常用的测试存储性能软件存在的缺陷，如测试费用昂贵、测试结果失真、应用场景限制、用户体验差等，以及用户对存储产品的吞吐量、响应时间、支持并发数等常规性能指标，及其用于具体业务领域的性能表现的要求越来越高的问题，进行了自治组件架构的存储业务仿真测试。首先分析了相关主流的性能测试软件的技术思路和软件架构，提出了自治组件架构的业务仿真性能测试软件的系统模型，并论述了其特点，然后阐述了该系统的基础设施与业务组件的设计和实现原理。

参 考 文 献

[1] Scholz-Reiter B, Kolditz J, Hildebrandt T. Engineering autonomously controlled logistic systems[J]. International Journal of Production Research, 2009, 47(6): 1449-1468

[2] 冯斌，赵雷，杨季文. 一个数据库老化测试工具的设计与实现[J]. 计算机应用与软件，2009，26(1)：153-155

[3] Scholz-Reiter B, Görges M, Jagalski T, et al. Modelling and analysis of autonomously controlled production networks[C]// Proceedings of the 13th IFAC Symposium on Information Control Problems in Manufacturing (INCOM 09). Moscow, Russia, 2009: 850-855

[4] 侯东风，刘青宝，张维明，等. 一种适应性的流式数据聚集计算方法[J]. 计算机科学，2010，37(3)：152-169

[5] 李雅红，魏必凡，王建国. 基于 J-sim 的主动节点带宽资源管理[J]. 计算机工程与设计，2008，29(14)：3582-3586

[6] Rekersbrink H, Makuschewitz T, Scholz-Reiter B. A distributed routing concept for vehicle routing problems[J]. Logistics Research, 2009, 1(1): 45-52

[7] Scholz-Reiter B, Sowade S, Rippel D, et al. A Contribution to the Application of Autonomous Control in Manufacturing[J]. International Journal of Computers, 2009, 3(3): 279-291

[8] 朱怡雯，钱超，林勇. 软件性能测试工具综述[J]. 中国金融电脑，2009(7)：79-82

[9] Hennessy J L, Patterson D A. Computer Architecture: A Quantitative Approach (4th Edition) [M]. Morgan Kaufmann, 2006: 15-16

[10] 王良，粟跃鹏，杨政. TPC-H 自动测试工具 TPCHDriver 的研究[J]. 计算机工程，2009，35(2)：18-20

[11] 黄锋，吴华瑞，朱华吉，等. Web 应用性能测试工具研究与实现[J]. 计算机工程与设计，2008，29(13)：3465-3467

[12] Scholz-Reiter B, Höhns H. Selbststeuerung logistischer Prozesse mit Agentensystemen[M]// Grundlagen, Gestaltung, Konzepte, G. Schuh, eds. Produktionsplanung und-steuerung. Berlin: Springer Verlag, 2006: 745-780

[13] 温艳冬. 软件性能测试需求的获取方法综述[J]. 软件工程师，2010(Z1)：124-127

[14] 蔡虹，孙志行，倪卫红. 电信行业软件的性能测试与性能调优[J]. 内江科技，2010(2)：107-108

[15] Ochoa O. Towards a Tool for Generating Aspects from MEDL and PEDL Specifications for Runtime Verification[J]. Lecture notes in Computer Science, 2007, 4839: 75-86