

[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 15/16 (2006.01)



[12] 发明专利申请公布说明书

[21] 申请号 200880012953.2

[43] 公开日 2010 年 3 月 3 日

[11] 公开号 CN 101663660A

[22] 申请日 2008.4.15

[21] 申请号 200880012953.2

[30] 优先权

[32] 2007.4.26 [33] US [31] 11/740,556

[86] 国际申请 PCT/US2008/004866 2008.4.15

[87] 国际公布 WO2008/133818 英 2008.11.6

[85] 进入国家阶段日期 2009.10.21

[71] 申请人 国际商业机器公司

地址 美国纽约

[72] 发明人 C·多莱 R·E·斯托姆

[74] 专利代理机构 中国国际贸易促进委员会专利
商标事务所
代理人 赵冰

权利要求书 4 页 说明书 19 页 附图 8 页

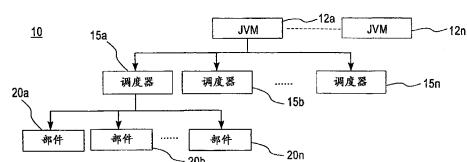
[54] 发明名称

分布式、容错和高可用性的计算系统

[57] 摘要

一种用于实现高可用的、容错地执行分布式计算系统中的部件的方法和系统，而不需要这些部件的编写者明显地编写代码（诸如，实体 beans 或数据库事务）来使得部件状态为持续的。它是通过把分布系统的固有的非确定性行为变换成确定性行为，从而通过有利的有效的检验点重新播放技术而达到状态恢复。该方法包括：调整执行环境，以便能够在部件之间进行消息传送；在程序执行期间自动把确定性时间戳与从发送器部件传送到接收机部件的消息相关联，所述时间戳代表所述消息到达接收机部件的预计时间。在部件处，在程序执行期间跟踪部件的状态并周期地检查在本地存储装置中的状态。在机器发生故障后，通过恢复最近存储的检验点并重复执行自从上次的检验点以来发生的事件而恢复部件状态。所述系统通过以与它们的相关联的

时间戳相同的次序处理所述消息重复进行所述接收部件的执行而具有确定性。



1. 一种计算系统，包括：

多个软件部件，每个实现执行任务的逻辑，在执行环境下执行的部件包括在网络中连接并适用于在部件之间传送消息的一个或多个机器；

用于在程序执行期间自动把确定性时间戳与从发送器部件传递到接收机部件的消息相关联的装置，所述时间戳代表所述消息到达接收机部件的预计时间；

用于通过使用所述时间戳确定性地执行所述部件以生成输入消息的唯一到达次序的装置；

用于在程序执行期间跟踪部件的状态和检查本地存储装置的所述状态的装置；

其中在机器发生故障后，通过恢复最近存储的检验点并重复执行自从上次的检验点以来发生的事件而恢复部件状态。

2. 如在权利要求 1 中所述的确定性计算系统，其中所述部件状态对于备份处理器装置被检查。

3. 如在权利要求 1 中所述的确定性计算系统，其中所述执行环境包括单个 Java 虚拟机，每个所述部件被存储在所述单个 Java 虚拟机中。

4. 如在权利要求 1 中所述的确定性计算系统，其中所述执行环境包括多个 Java 虚拟机，所述部件以分布式方式在所述多个 Java 虚拟机之间执行。

5. 如在权利要求 4 中所述的确定性计算系统，其中所述执行环境包括用于管理一个或多个部件的执行的调度器部件，所述调度包括为在所述执行环境中的部件分配执行线程。

6. 如在权利要求 4 中所述的确定性计算系统，其中部件从多个原有部分部件接收输入消息，所述系统还包括合并器装置，用于根据输入消息的时间戳执行确定性合并。

7. 如在权利要求 1 中所述的确定性计算系统，其中所述部件的状态被间歇地更新。

8. 如在权利要求 1 中所述的确定性计算系统，其中所述时间戳作为消息到达所述接收部件的真实时间的估计被计算。

9. 如在权利要求 1 中所述的确定性计算系统，还包括：

用于检测在来自部件的连续的消息传输之间的时间间隙的装置；以及

能够发送寂静消息的装置，用于保证来自所述执行环境中的所述部件的正确的时间消息流。

10. 如在权利要求 9 中所述的确定性计算系统，还包括：

用于当部件接收机确定由于在机器之间的网络连接的故障，某些消息或寂静消息已丢失时，把好奇消息发送回发送器部件以触发重新发送任何以前的消息或寂静消息的装置。

11. 一种用于确定性执行在提供执行环境的计算系统中的部件的方法，所述计算系统适用于能够在所述部件之间进行消息传送，每个所述部件实现执行任务的逻辑，所述方法包括：

在程序执行期间自动把确定性时间戳与从发送器部件传送到接收机部件的消息相关联，所述确定性时间戳代表所述消息到达接收机部件的预计时间；

通过使用所述时间戳确定性地执行所述部件以生成输入消息的唯一到达次序；

在部件处，在程序执行期间跟踪该部件的状态并检查在本地存储装置中的所述状态；

其中在部件发生故障后，通过恢复最近存储的检验点并重复执行自从所述上次的检验点以来发生的事件而恢复所述部件状态。

12. 如在权利要求 11 中所述的方法，还包括：通过以与被记录在所述存储的状态中的所述时间戳相关联的消息相同的次序处理所述消息而重复进行所述接收部件的执行。

13. 如在权利要求 11 中所述的方法，还包括按照与接收到的消

息相关联的所述时间戳，管理在所述执行环境中的所述一个或多个部件的执行，所述管理包括为在所述执行环境中的部件分配执行线程。

14. 如在权利要求 13 中所述的方法，其中部件从多个原有部分部件接收输入消息，所述方法还包括根据输入消息的时间戳执行确定性合并。

15. 如在权利要求 13 中所述的方法，还包括间歇地更新所述部件的软检验点。

16. 如在权利要求 11 中所述的方法，其中所述时间戳作为消息到达所述接收部件的真实时间的估计被计算。

17. 如在权利要求 11 中所述的方法，还包括：

在部件处检测在来自所述部件的连续的消息传输之间的时间间隙；以及

使能发送寂静消息，用于保证来自所述执行环境中的所述部件的正确的时间消息流。

18. 如在权利要求 17 中所述的方法，还包括：

当部件接收机确定某些消息或寂静消息已丢失时，把好奇消息发送回发送器部件以触发重新发送任何先前的消息或寂静消息。

19. 一种用于使能确定性执行面向部件的应用的系统，包括：

使能在执行环境中的部件之间进行消息通信的子系统，所述部件包括一个或多个端口，用于从所述部件接收消息输入或从所述部件输出消息，每个所述部件实现执行任务的逻辑；

用于把高级别设计映射到低级别实施方案的装置，其中部件被分配给所述执行环境的特定处理引擎；

代码增强装置，用于：

用在程序执行期间的时间戳和发送器部件传送到接收机部件的消息一起，自动扩充相关联的部件的输入或输出消息，所述时间戳代表所述消息到达接收机部件的预计时间；以及

用用于跟踪部件的状态的增量改变的代码来扩充部件；

其中在程序执行期间跟踪该部件的状态，所述部件状态在本地存

储装置中被检查，以及

其中在部件发生故障后，通过恢复最近的检验点并重复执行自从上次的检验点以来发生的事件而恢复部件状态。

20. 如在权利要求 19 中所述的系统，其中所述执行环境是分布式计算系统。

21. 一种由机器可读的程序存储装置，有形地体现可由机器执行的指令的程序，执行用于确定性执行在计算系统中的部件的方法步骤，所述系统适用于使能在所述部件之间进行消息传送。每个所述部件实现执行任务的逻辑，所述方法步骤包括：

在程序执行期间自动把确定性时间戳与从发送器部件传送到接收机部件的消息相关联，所述确定性时间戳代表所述消息到达接收机部件的预计时间；

通过使用所述时间戳确定性地执行所述部件以生成输入消息的唯一到达次序；

在部件处，在程序执行期间跟踪部件的状态并检查本地存储装置中的所述状态；

其中在部件发生故障后，通过恢复最近存储的检验点并重复执行自从所述上次的检验点以来发生的事件而恢复所述部件状态。

22. 如在权利要求 21 中所述的由机器可读的程序存储装置，其中所述系统通过以与被记录在所述存储的状态中的所述时间戳相关联的消息相同的次序处理所述消息重复进行所述接收部件的执行而具有确定性。

分布式、容错和高可用性的计算系统

技术领域

[0001]本发明总体上涉及用于开发和执行分布式系统的应用开发工具、方法和系统，更具体地说，涉及用于开发和执行用于执行面向部件的应用的分布式、透明容错和高可用性的系统的改进环境。

背景技术

[0002]SCA 服务部件体系结构提供开放的、技术中立的模型，用于实施定义商业功能的 IT 业务。SCA 还提供用于从各种服务的集合收集商业解决方案的模型，其中对于诸如接入方法和安全性的解决方案的方面进行控制。通过 SCA，消费者能够更容易地创建新的 IT 资源并把现有的 IT 资源转换成可重复利用的业务，它们可以快速地适应日益改变的商业需求。能够构建 SCA(中间件)部件的技术规范利用面向服务的体系结构(SOA)，SOA 把 IT 资源构建成执行商业功能的一系列可重复利用的业务。面向服务的体系结构将调用远程目标和功能(被称为“服务”)的能力与用于动态服务发现的工具组合到一起，重点放在互操作性上。当前，该行业的目标是给应用开发者提供基于 SOA 构建应用的更简单且更强有力的方式。

[0003]此外，在开发实施 SCA 部件的分布式系统时，一个目标是提供“非易失性”数据的透明的和容错的可用性，非易失性数据可以代表以容错方式预留的永久“设置”(将被存储在分布式系统的大容量媒体中)或“状态”。现在，分布式容错和高可用性的系统的开发是专用的、易于出错的和费时的。当前的解决方案类似于示例的货币汇兑系统，其中货币价格的波动和汇兑操作可能是无序的或非原子性 (non-atomic) 的。由于网络或线程 (threading)，执行通常是非确定性的：现有的用于持久性的机制(实体 beans, JDBC 等等)是特别重

要的，它们需要额外的知识和额外的代码。

[0005]例如，当前的解决方案实现实体 beans，例如“Enterprise Java Bean”(EJB)，它包括用于 J2EE 平台的服务器侧部件体系结构。EJB 意图支持快速和简化地开发分布式、事务性、安全和便携的 Java 应用。EJB 支持允许消息同时消费的容器体系结构，并且 EJB 为分布式事务提供支持，这样，数据库更新、消息处理和通过使用 J2EE 体系结构到企事业系统的连接可以加入同一事务上下文。

[0006]特别希望消除需要程序员学习专门的方法和结构，诸如事务、JDBC、或把部件状态分离成分开的对象并持续该状态的实体 beans，而代之以为普通代码自动提供持久性和容错(也称为透明容错)。

[0007]确实已有用于分布式系统中的透明容错技术，包括在由本发明的受让人共同所有的美国专利 No.4,665,520 中描述的技术。这种技术的性能受到在分布式系统中的通信部件的性能的非确定性的限制，因为从一个分布式的部件到另一个部件的每次通信均需要被记录。

[0008]而且，特别希望提供透明地支持确定性执行、容错和高可用性的执行服务器，以避免恢复非确定性分布性系统的性能问题。

[0009]另外，特别希望为程序员提供基于简单的基于部件的模型，具体来说，希望提供用于使得中间件功能对于应用开发者更容易接入的系统和方法。

发明内容

[0010]因此，本发明的广义目的是克服如上所述的现有技术的缺点。

[0011]本发明的另一个目的是为面向部件的应用提供透明地支持确定性执行、容错和高可用性的执行环境。

[0012]这些目的以及其他相关的目的的完成是通过计算系统和方法实现的。该确定性计算系统包括：

多个软件部件，每个实现执行任务的逻辑，在执行环境下执行的部件包括在网络中连接并适用于在部件之间传送消息的一个或多个机器；

用于在程序执行期间自动把确定性时间戳与从发送器部件传送到接收机部件的消息相关联的装置，所述时间戳代表所述消息到达接收机部件的预计时间；

用于通过使用所述时间戳确定性地执行所述部件以生成输入消息的唯一到达次序的装置；以及

用于在程序执行期间跟踪部件的状态并周期地检查在本地存储装置中的状态的装置；

其中在机器出现故障后，通过恢复最近存储的检验点并重复执行自从上次的检验点以来发生的事件而恢复部件状态。

[0013]计算系统通过以与它们的相关联的时间戳相同的次序处理消息而重复执行接收部件从而具有确定性。

[0014]除了本发明的这个方面以外，部件状态可以对于备份处理器装置被检查。

[0015]按照本发明的另一个实施例，提供了用于确定性执行在计算系统中的部件的方法，所述计算系统适用于能够在所述部件之间进行消息传送，每个部件实施执行任务的逻辑，该方法包括：

在程序执行期间自动把确定性时间戳与从发送器部件传送到接收机部件的消息相关联，所述时间戳代表所述消息到达接收机部件的预计时间；

通过使用所述时间戳确定性地执行所述部件以生成输入消息的唯一到达次序；以及

在程序执行期间跟踪该部件的状态并周期地检查本地存储装置的状态；

其中在部件出现故障后，通过恢复最近存储的检验点并重复执行自从上次的检验点以来发生的事件而恢复部件状态。

[0016]按照本发明的另一个实施例，提供了用于执行面向部件的

应用的环境，所述环境适用于在所述部件之间传送消息。该环境包括：

能够进行包括端口的技术规范的部件高级别设计的子系统，所述端口代表到部件的消息输入和来自部件的消息输出，每个部件实现执行任务的逻辑；

布局服务，用于把高级别设计映射到低级别实施方案，其中部件被分配给所述执行环境的特定的处理引擎；

代码增强装置，用于：

通过在程序执行期间的时间戳和从发送器部件传送到接收机部件的消息一起，自动扩充相关联的部件的输入或输出消息，所述时间戳代表所述消息到达接收机部件的预计时间；以及

通过用于跟踪部件状态的递增改变的代码来扩充部件；

其中在程序执行期间跟踪该部件的状态并检查在本地存储装置中的部件状态；以及

其中在部件出现故障后，通过恢复最近的检验点并重复执行自从上次的检验点以来发生的事件而恢复部件状态。

[0017]按照本发明的这个方面，高级别设计是相对于在所述执行环境中部件在何处执行无关的。

[0018]特别希望提供按照所描述的每个实施例的分布式系统，使得两种类型的部件能够共存：非时间感知的部件，其中对时间的估计是自动的；和时间感知的部件，其中程序员可以规定实时约束条件。

[0019]本领域技术人员在研究附图和详细说明后将了解本发明的另外的优点。并且打算在这里引用任何附加优点。

附图说明

[0020]通过与附图相组合做出的以下的详细说明，本领域技术人员将清楚本发明的目的、特征和优点。

[0021]图 1 示出了根据本发明的执行服务器体系结构 10，其中运行中间件应用，用于执行部件和面向部件的应用；

[0022]图 2 示出了在单个执行引擎-本例中为 Java 虚拟机(JVM)

中的部件之间、以及在按照本发明的执行服务器体系结构的多个执行引擎之间的示例性信息传送和通信；

[0023]图 3A-3D 示出了能够由本发明的服务器中间件部件使能的示例性应用，包括从示例性用户设计的部件布局(图 3A, 3B)，虚拟时间估计(图 3C)，和寂静与好奇消息生成(图 3D)；

[0024]图 4 示出了按照本发明的原理的对存储器存储装置的检查方法和从硬盘恢复检验点的概况；

[0025]图 5A-5C 示出了使用被动远端备份和生产调度器复制品以用于提供由本发明给予的高可用性的概况。

具体实施方式

[0026]如上所述，所提出的发明目的是解决现有技术中的问题，即，不断地需要给程序员提供专门的方法和结构，诸如事务、JDBC、或把部件状态分离成分开的目标的实体 beans，以便持续该状态，以及在非确定性实施方案中需要记录在部件之间的消息。这是通过提供确定性的和高可用性的执行服务器而解决的，所述执行服务器自动提供用于执行面向部件的应用的持久性和容错。

[0027]图 1 示出了执行服务器体系结构 10，其中存在的中间件应用被运行，以便执行部件和面向部件的应用，总体上表示为部件 20a,...,20n。如图 1 所示，每个服务器包括一个或多个 Java 虚拟机，总体上表示为 JVM's 12a,...,12n，诸如图 1 所示。应该理解，可以有多个服务器，并且在每个服务器内，可以有多个 Java 虚拟机(JVM)实例，例如作为在单个机器内不同的处理过程运行。应该理解，Java 和 JVM 的使用纯粹是示例性的；除了 JVM，也可以使用藉以编写应用部件的其他语言的执行环境。在每个 JVM 内提供了调度器层，包含至少一个调度器，总体上表示为调度器 15a,...,15n，作为在 JVM 与分级结构中的、对于开发者透明的部件之间的层。调度器 15a,...,15n 是在 JVM 中一个或多个密切相关的部件连同决定何时执行这些部件的逻辑的分组。在 JVM 中可以有多个调度器，每个管理它自身的相

应部件的执行。例如，如图 2 所示，JVM 12 实现用于管理部件 21 和 22 的执行的调度器 15，并实现用于执行部件 26, 27 的调度器。第二 JVM 13 包括用于管理部件 28 和 29 的调度器 17。具体地，如图 2 所示，在 JVM 与分级结构中的部件之间的调度器层管理 JVM 中的部件的执行。应该理解，在 JVM 内，在部件之间的通信和在调度器之间的通信通过参考传递有效地实现。在 JVM 中可以有多个调度器，每个管理它自己的部件的执行。当部件具有待决的输入消息时，调度器将选择适当的时间来分配可用的线程以执行它。这个决定是根据多个因素作出的，诸如系统负荷、用户需要的吞吐量、由排队造成的存储器消耗等等。当调度器决定同时执行多个部件时得到同时性。

部件

[0028]正如已知的，部件可以是面向服务的或面向事件的，以及可以是遵从如 SCA 那样的部件模型的“对象”的任何集合。典型地，Java 语言或 C++ 语言或类似的面向对象的语言，或其他语言，诸如 Python 或 Perl，被使用于实现 SCA 服务部件和在部件之间发送的数据。也就是说，在部件之间的交互仅仅通过在端口之间传递数据消息，或通过从服务消费者到服务供应商的服务呼叫而发生，其中数据值可被传递和返回。发送的消息或服务呼叫的接收者通过同步的方法表现为对象。在部件内在任何时间存在控制一个线程。在部件之间从来没有共享的对象。对象是以下的任一项：(a) 部件对象本身，通过同步方法的“监视器”，(b) “数值”对象，其可以从部件传递到部件，但从不共享，或(c) “实现”对象，其可以被共享，但仅仅在部件对象本身内或相同的数值对象内。这个规定尤其保证了没有数据被一个以上的执行部件同时拥有，并在作者为 David Bacon, Robert Strom, Ashis Tarafdar, 标题为“Guava: a dialect of Java without data races,” Proceedings of the 15th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, 2000 的参考文献中被规范化并加以描述，其内容和公开内容整体地在

此引用以供参考，就如同这里详细阐述的一样。按照 Guava 技术，有可能统计地检验特定的部件服从这个规定。在 JVM 上运行的 Java 的 Guava 方言仅仅是适合于在本发明的环境中使用的那种部件实施语言的一个例子。

[0029]为了使得部件“能够连线（wireable）”，用于由部件提供服务的服务呼叫的输入方法，或用于由部件接收的异步消息的消息队列，作为输入“端口”使其从外部可获得。进行呼叫或发送消息到其它部件的站点作为输出“端口”使其从外部可获得，如图 2 所示。假设作出分开的部署-时间判决，确定(a)输出端口如何被“连线”到输入端口，和(b)在哪里放置部件。由于这些判决的结果，某些端口连接可以是本地的，而其他连接是远端的。应该理解，当端口连接连接同一个 JVM 内的部件(例如图 2 上的连接 40)时，消息和服务呼叫可以更有效地传输(例如，“通过参考”), 以及当这些部件在不同的 JVM 中(例如图 2 中的连接 42)时，不太有效地传输(例如通过复制或通过发送网络消息)

[0030]如所提到的，本发明的执行服务器被设计成支持跨机器的分布式执行。Java 虚拟机(JVM)的多个实例可以在那些机器上运行或作为在单个机器内的不同的处理过程运行，取决于部署。如图 2 所示，用于在 JVM 之间的消息通信的示例性协议是用户数据报协议(UDP) 30。因此，在如图 2 所示的示例性实施例中，JVM 12 经由 UDP 消息传递协议与 JVM 13 通信。正如已知的，UDP 的使用并不提供保证的传递，但在传输控制协议(TCP)下表现出更好的性能。作为替代，在本发明的中间件部件中构建不丢失的和保持次序的消息传递，这将在下面更详细地描述。

开发

[0031]本发明的执行服务器不同于其他手段，因为：(1)开发环境基本上被简化：部件可以以普通的 Java 编写，并且可以被连线，如在 SCA 部件模型中那样，以构建分布式的流网络；(2)部件以普通的 Java 原语变量和集合类别存储状态；(3)程序员不需要使用专门的方

法和结构，诸如事务、JDBC、或把部件状态分离成分开的对象并持续该状态的实体 beans。作为替代，由本发明的执行服务器自动提供持久性和容错，尽管出现故障，以及出故障的网络可能丢弃、重新排序或复制消息。

确定性执行

[0032]按照本发明的方法，用于容错的技术是基于保证系统的确定性的、可重放的执行。这是对于基于事务、基于复制、或基于永久记录所有的部件间消息的其他方法的改进。确定性执行意味着，如果系统被给予相同的状态和相同的输入消息，则它将生成相同的输出消息。确定性是指在部件发生故障时，部件的状态可以通过恢复最近的检验点和回放自从该检验点以来发生的事件而被恢复。由于确定性，在回放后的状态保证是与丢失的状态相同的。这意味着，在状态每次被更新时不需要永久地保存该状态，而仅仅间歇地保存。

[0033]确定性执行的实现是基于从离散事件模拟概括出的技术，并把它们应用到执行服务器的运行时间环境。正如在事件模拟器中，在部件之间通信的每个消息被加上虚拟时间(VT)标记。和事件模拟器不同的是，虚拟时间被选择为消息将到达接收部件的真实时间的确定性估值。本发明的执行服务器保证系统行为就好像消息已经以虚拟时间次序到达那样。虚拟时间与真实时间之间的对应性越好，系统将执行得越好；然而，不管这种对应性如何好，都保证确定性执行。在这种方法的变体中，虚拟时间是这些真实时间截止期，并且系统将调整调度以保证满足这些真实时间截止期。这和事件模拟器的情况也不同，其中模拟的虚拟时间可能与真实时间没有明显的相关性。

[0034]确定性执行可以通过用包含虚拟时间(VT)的时间戳扩充所有的通信而实现。在系统边界处，接收外部事件，在它们被生成时它们不包含时间戳。按照本发明，系统无需人工干预地自动分配 VT 给那些事件。VT 符合因果次序，即，如果输出是由输入引起的，则它在早先的虚拟时间不一定发生，它是真实时间的近似。记录表记录所有分配的 VT，因为它们的生成是非确定性的，并且当在以后需要

回放时，记录表将是必不可少的。一旦事件在边界处用 VT 进行扩充，它们经由在部件之间的连接传输，并且以确定的次序处理。不需要将来的登记。这是与由最佳恢复和其他现有的透明容错技术所使用的方法不同的，其中需要记录在部件之间的消息，以使得非确定性到达次序可被重放。应该理解，没有两个消息具有相同的 VT。

[0035]当部件被选择以执行时，它可以产生输出作为处理输入消息的结果。输出与严格地大于输入消息的 VT 的 VT 相关联，反映非零计算延时。在 VT 中的增量以及输出消息完全由输入消息决定。部件例如可以从多个原有部分的部件接收输入，发送消息，或提供呼叫消息等等。在这种情况下，根据消息的 VT 利用确定性合并。VT 由原有部分独立地生成，但它们在合并时本地地进行比较。一旦已知在将来没有原有部分可以以早先的 VT 发送任何消息，就可以安全地处理具有最小 VT 的消息。应该理解，在替换实施例中，消息可以积极地处理，并且如果具有早先 VT 的消息到达，则消息被退回重来，正如参考 Jefferson, D 的“Virtual time”，ACM Transaction on Programming Languages and System, July 1985 中描述的离散事件模拟环境中那样。因为确定性 VT 生成和确定性合并(二者对于部件都是本地的)，消息处理的次序是确定性的。

部署时间机制

[0036]由于为在用于持续分布式计算的确定性执行机制之间的主要差别，在本发明的方法中只需要非常少的用户干预。具体地说，不需要开发者知道 VT 或如何计算它们。提供了一组动态地简化应用开发者的工作的工具，而这里描述的好处完全不受损害。

布局服务

[0037]如图 3A 所示，在设计时，部件的计算逻辑可以以普通 Java, C++ 或类似的面向对象的代码，或以诸如 Python 或 Perl 的其他语言编写。端口例如用 Java 接口定义。逻辑连接由设计者通过连线两个或多个端口创建。这个高级别设计完全抽象出最终系统的分布

特性。它也认可诸如确定性执行和容错这样的特性。如图 3B 所示，中间件布局服务部件 50 被用来把诸如图 3A 所示的设计那样的这种高级别设计映射到较低级别的实现，其中大多数互相关联的部件被分组成调度器，调度器被分配到可能在不同机器上的 JVM。在执行之前的布局是半自动的，具有用户提供的参数，诸如机器的计算能力、网络容量、和布局优选项的量度。为了使得性能最大化，平衡不同机器上的工作负荷。在执行时，初始布局可以通过把调度器从它们的原先的机器移到另外的机器而被精细调节，以便得到甚至更好的总体性能。根据布局，从开发者的观点看来，逻辑连接也可以被映射到物理连接。

[0038]该布局服务器 50 简化部署任务，但通过接受用户规定的配置而仍旧提供足够的灵活性。

[0039]因此，在现在给出的用于示例目的的布局的一个非限制性例子中，如图 3A 所示，显示了售票系统 30 的示例性设计者的高级别方法的图，售票系统 30 可被用来给来自两个不同的请求路径的消息的接收建模，即例如请求者 32, 34 请求在事件中预留座位。这些门票请求首先被接收，然后被处理以便确定由每个请求者请求的门票的总数(例如 5 张门票或 10 张门票)，然后，最后分配给用于在打印机设备处打印门票的过程。以下的示例性代码部分描述部件的逻辑(例如，原语变量、集合类别等等)，可以按照图 3A 描述的示例性系统的高级别设计以普通 Java 编写。

```

class Requester extends Component {
    Requester(Scheduler s, String ID) { ... }
    public RequestPort requestOut = ...
}
class TopOfN extends Component {
    TopOfN(int n, Scheduler s, String ID) { ... }
    public RequestPort requestIn = ...
    public RequestPort requestOut = ...
}
class Assigner extends Component {
    Assigner(Scheduler s, String ID) { ... }
    public RequestPort requestIn = ...
    public PrintPort assignmentOut = ...
}
class Printer extends Component {
    Printer(Scheduler s, String ID) { ... }
    public PrintPort println = ...
}

```

[0040]布局服务 50 将生成如图 3B 所示的用于低级别实施方案

的代码。在如图 3B 所示的最终的低级别实施方案中，通过规定用于接收消息的第一 JVM 62 的端口，对请求者进行建模；并对在第二 JVM 64 处规定的相应端口进一步建模，第二 JVM 64 处理在分配者部件 65 处的相应的请求并合并请求，部件 65 分配打印机功能部件，通过规定用于包括执行安排的门票打印过程的打印机部件 67 的第三 JVM 66 的单个端口，对打印机功能部件进行建模。以下的示例性代码部分描绘布局逻辑，该布局逻辑可以按照图 3B 所示的示例性系统的低级别设计以普通的 Java 编写。以下的示例性代码描绘第一 JVM 62 的配置，如图 3B 所示的第一 JVM 62 被设计为包括第一调度器 72，用于管理请求者部件 32，34。

```
class JVM1_Placement {
    public static void main(String[] args) {
        Scheduler s1 = new Scheduler(0, 1000);
        Requester r1 = new Requester(s1, "r1");
        Requester r2 = new Requester(s1, "r2");
        RequestPort t5In = (RequestPort)
            s1.createRemoteOutputPort(
                RequestPort.class,
                "localhost:1001/t5/requestIn", 0);
        s1.connect(r1.requestOut, t5In);
        RequestPort t10In = (RequestPort)
            s1.createRemoteOutputPort(
                RequestPort.class,
                "localhost:1001/t10/requestIn", 1);
        s1.connect(r2.requestOut, t10In);

        s1.start();
    }
}
```

[0041] 以下这个示例性代码描绘第二 JVM 64 的配置，如图 3B 所示的第二 JVM 64 被设计为包括调度器 74，用于管理对应于请求者部件 32，34 的处理部件。

```

class JVM2_Placement {
    public static void main(String[] args) {
        Scheduler s2 = new Scheduler(2, 1001);
        TopOfN t5 = new TopOfN(5, s2, "t5");
        TopOfN t10 = new TopOfN(10, s2, "t10");
        Assigner a = new Assigner(s2, "a");
        s2.register(0, t5.requestIn);
        s2.register(1, t10.requestIn);
        s2.connect(t5.requestOut, a.requestIn);
        s2.connect(t10.requestOut, a.requestIn);
        PrintPort pIn = (PrintPort)
            s2.createRemoteOutputPort(
                PrintPort.class,
                "localhost:1002/p/println", 0);
        s2.connect(a.requestOut, pIn);
        s2.start();
    }
}

```

[0042]以下这个示例性代码描绘第三 JVM 66 的配置，如图 3B 所示的第三 JVM 66 被设计为包括调度器 76，用于管理来自单个端口的打印处理部件。

```

class JVM3_Placement {
    public static void main(String[] args) {
        Scheduler s3 = new Scheduler(1, 1002);
        Requester p = new Printer(s3, "p");
        s3.register(0, p.println);
        s3.start();
    }
}

```

自动代码增强器

[0043]在本发明中，程序变换器扩充用户编写的部件，以产生与运行时间执行环境兼容的部件。具体地，程序变换器提供以下的增强：1) 输入和输出消息或方法调用的所有的接口用保持 VT 的字段被扩充；2) 用估值器扩充处理到达输入端口的消息的每个方法，估值器计算针对它生成的每个输出消息和针对从该方法的返回值的“VT 差值”。“VT 差值”代表从方法的开始到生成的输出消息或返回值所过去的真实时间量的确定性估计；以及 3) 每个部件用跟踪自从上一个软检验点以来它的状态的增量改变的代码扩充，并在从调度器请求时将增加的软检验点记录串行化。

[0044]软检验点之所以这么称谓是因为可以丢失任何单个检验点而不牺牲系统的恢复能力-丢失的检验点仅仅意味着必须从以前的

检验点进行恢复，这可能延长在故障后恢复的时间，但不会影响最后的恢复能力。相反，部件用从增加的软检验点记录进行重建状态的代码被扩充。可选地，部件可以用生成“渴望寂静”的代码扩充。也就是说，假定现在已知在给定的时间 t 在它的输入端口上没有输入消息到达，它计算在给定的输出端口上有可能出现消息的时间 t 之后的最早的 VT 差值。这样的计算可被用于将寂静发送到连接到这些输入端口的部件。由于一系列寂静时间戳承诺不再有消息从该部件用那些时间戳发送，这样的信息有可能使得接收部件能够接着进行处理等待消息，因为它现在被认为是最早可能的消息。

[0045]图 3C 更详细地描绘用于扩充消息结构的 VT 的确定方法，对于实施本发明所需要的消息结构正如在图 3B 的例子中描述的示例性售票系统中低级别设计显示的。应该理解，消息或事件(“e”)与在系统边界上的 VT 相关联。当前的 VT 以遵从因果关系的任意方式生成。然而，它们可以是真实时间的估计。因此，如图 3C 所示，在运行期间，对于包括在第一 JVM 62 的第一请求者部件 32 处接收的示例性相关的数据串(e_1^1)的所接收的消息结构 42，有被添加到消息结构 42 的第一 VT 值(vt_1^1)与其相关联，而包括在第二请求者 34 处接收的示例性相关的数据串(e_1^2)的消息结构 44 与被添加到消息结构 44 的 VT 值(vt_1^2)相关联。图 3C 还描绘了对于自从上一个软检验点以来它的状态的增量改变的跟踪，在从调度器请求时这导致生成由进入数据存储装置中的记录 99 描绘的增加的软检验点记录。这个记录 99 记录所有分配的 VT，因为它们的生成是非确定性的，并且在需要重放时这个记录是必不可少的，正如这里在下面更详细地描述的。这些扩充的事件或请求消息 42($e_1^1; vt_1^1$)和 44($e_1^2; vt_1^2$)，每个在它们调度的 VT(虚拟时间)时被输入到第二 JVM 64，在那里它们由在 JVM 64 中相应的请求消息处理部件处理。也就是说，如图 3C 所示，在第二 JVM 64 中处理接收的输入扩充请求消息 42 后，生成了另一个消息 52，消息 52 被扩充成包含从第二 JVM 64 的第一部件输出的计算的 VT 值($e_2^1; vt_2^1$)；同样地，生成了另一个消息 54，其被扩充成包含从第二 JVM 64 的第

二部件输出的计算的 VT 值($e_2^2; vt_2^2$)。在合并点，如由分配器部件 65 描绘的，来自多个流的事件通过使用它们的 VT 进行比较。因此，如图 3C 所示，来自第二 JVM 的分配器部件是确定性地合并的消息序列 55，包括按时间排序的消息 52 和 54，即， $[e_2^1; vt_2^1]$ 和 $[e_2^2; vt_2^2]$ 。在第三 JVM 66 中实现的确定性合并器然后将根据在接收到的消息 55 中的合并的 VT 值以对于打印部件 67 的确定性次序排序操作。

消息丢失的容忍度和重新排序

[0046] 如上所述，在 JVM 内，在部件之间的通信通过参考传递有效地实现。JVM 内通信也是无丢失的和保持次序的。在 JVM 之间，可以使用 UDP 通信，虽然这可能引起消息丢失或重新排序，因为这样的丢失或重新排序是本发明的中间件部件容忍的，实现寂静的概念以及好奇 (curiosity) 消息在调度器之间但不在调度器内发送。

[0047] 继续参照图 3D，图中显示加入寂静和好奇消息，以保证在本发明的系统中丢失消息的检测。正如在图 3A-3C 上描绘的和现在在图 3D 中显示的本发明的示例性应用中提到的，假设在 VT 中存在时间间隙，也就是说，除非两个消息在时间线上互相精确地相邻。时间间隙的存在由服务器的中间件部件、例如调度器检测。例如，在第一 JVM 62 发送第一被扩充的消息 42 后和在发送第二被扩充的消息之前，JVM 62 生成和发送一个或多个寂静，即寂静消息 81，来填充间隙，以使得接收机、例如 JVM 64 连续地得知消息或寂静。寂静消息规定一对数目，指定被认为不包含消息的一系列 VT 的时间点。例如，正好在消息 42 之前发送的、在图 3D 中的消息 80 可包含数据 [Silence;100: vt_1^1-1]，意思是指在从 100 到 vt_1^1-1 的范围内 VT 的所有的点被认为不包含消息。由于寂静消息，接收机可以认为每个点肯定包含消息或寂静。如果数据或寂静消息丢失，则接收机将通知一个间隙。间隙检测在接收机一侧被执行，如果在某个时间间隔(被定义为用户可配置或可编程的参数)内存在间隙，则接收机、例如 JVM 64 可以判断某些消息或寂静已丢失，然后把用于接收的好奇消息 83 发送回

到发送者，例如 JVM 62。好奇消息规定一对数目，指定接收机对于其不知道数据或寂静从而对其“好奇”的一系列 VT 的时间“点”。例如，如果这个寂静消息由于某种原因而丢失，则好奇消息将读出 [Curiosity; 100: vt_1^1-1]。好奇消息将触发由发送者在被询问的点的范围内重新发送以前的消息或寂静。发送者必须准备好重新发送这些消息或寂静，直至接收机成功地取得软检验点，从而承诺绝不再次请求重新发送为止。

[0048]如果好奇消息丢失，或重新发送的消息丢失，则好奇消息将被重新发送。只要同一个消息不是无限地经常丢失，最终，丢失的消息将被找回。应该理解，消息可能较迟到达，以及可能复制重新发送的消息，但这不会引起任何问题，因为接收机丢弃其 VT 匹配于它已处理的消息的时间的任何消息是安全的。(没有两个消息具有相同的 VT)。重新排序也是被容忍的，因为接收机只拒绝处理如果在它之前仍旧有间隙的消息。因此，如图 3D 所示，排队、寂静和好奇一起保证消息以正确的次序处理。

虚拟时间估计器

[0049]至今为止，允许 VT 生成是任意的，只要它是完全确定性的并且它保留消息因果性。任何这样的 VT 分配保证确定性执行。然而，如果 VT 太远离与真实时间的同步，则将看到性能惩罚，因为在合并时，接收机将拒绝处理下一个消息，直至保证它在 VT 上较早先没有接收任何消息为止。这可能引起在真实时间上较早生成的消息在以后被处理，因为由于估值的不精确性，它们得到较大的 VT。自动 VT 估计器被提供来试图保持 VT 和真实时间近似同步，以使得这种悲观主义不会造成不可接受的性能损失。对于这种 VT 估计有静态(即，预先执行)和动态(即，在执行期间)部件。静态地，自动 VT 估计器根据代码的已知的复杂性估计在各个部件上的计算延时。在部署时，估计器还可考虑其上部署部件的环境的已知的特性。这将调节计算延时估值，以反映诸如网络延迟时间、平均 CPU 负荷、垃圾收集

器性能、预期的存储器消耗、线程等等。然而，应当强调指出，计算延时估值必须是部件状态的确定性的、可重复的函数。它可以不考虑非确定性因素，诸如实际的当前 CPU 负荷。如果不幸地，调度器通知在 VT 与真实时间之间的差异变得过大，则它可以采取以下两种可能的行动之一：

1. 调度器可以相对于在机器中的其他调度器改变它的优先级，以使得它减慢或加速，并减小差异。
2. 调度器可以做出非确定性判决-所谓的“确定性故障”-以调节参数到调度器内使用的估计器。这种调节是非确定性的，被允许考虑非确定性因素，不像通常的计算延时估计那样。因为这样的判决违反了确定性，这种行动必须被记录在稳定的存储装置中，以便保证正确的重新播放。实际上，在判决之前 VT 的任何重新播放必须使用旧的估计器，以及在判决之后 VT 的任何重新播放必须使用新的估计器。确定性故障可能导致改变对于特定消息的特定时间估计的改变。系统的目的是使得静态估计器足够好，使得在 VT 与真实时间之间的漂移最小化，并且可通过调节调度优先级而控制漂移，以使得确定性故障是极罕见的事件。

真实时间分析

[0050]对于非时间感知的部件，通过本发明的中间件的 VT 估计器，使得时间戳的引入自动进行。然而，某些应用可能需要知道时间；或者在甚至更复杂的情景下，某些应用可包含非时间感知部件和时间感知部件。为了满足对于定时控制的需要，本发明的中间件部件允许定时约束条件由设计者规定，这些约束条件被自动汇编成额外时间控制码。在运行时间，这个代码通过将部件按优先级排列或移动部件而使得调度器能够满足时间约束条件。也就是说，真实时间调度器可被编程为与布局服务一起工作，保证真实时间截止期。布局服务可能需要在部署这样的调度器之前进行某种许可控制，以保证有足够的保留容量来满足截止期。另外，可能出现新的确定性故障种类：如果由于

非正常条件，对于给定的计算看起来将不满足真实时间截止期，则可以采取非确定性判决，使用替代的计算(使用大概较少的开销)。对于任何确定性故障，需要记录作出这样的改变的判决。

具有高可用性的机器故障的容忍度

[0051]在分布式系统中，机器可能被关断或与网络非预期地断开连接，或可能出现故障。许多同时期的应用利用在机器之间的合作。多个机器的使用能够给出高可用性。通过这个特征，某些机器的故障不妨碍应用的总体功能。在本发明的执行服务器中，故障容忍是透明的。中间件部件间歇地创建用于各个调度器的软检验点。软检验点是调度器状态的紧凑的图像，包括在其中的部件，建立的连接，排队的消息，等待服务呼叫等等。软检验点可包含全部状态，或可包含自从以前的软检验点以来的增量改变。用户定义的部件的状态(例如，包括语言级实体，诸如通过交叉参考的 Java 原语值和目标)也被记录在检验点中，使得当机器发生故障并恢复时，计算可以继续进行。

[0052]因此，对于在图 3A-3D 中描绘的和现在在图 4 中显示的本发明的示例性应用，对于在 JVM 64、66 中的每个调度器的检验点信息 84、86 分别被存储和被间歇地或递增地更新至本地存储器存储装置，例如硬盘驱动 94、96。然而，应该理解，这些检验点也可被存储在远端机器上。

[0053]在执行期间，JVM 机器可能发生崩溃，由于这个事件，在其上运行的调度器停止并完全失去自从它们最后的检验点以来它们的状态。当机器重新启动时，它用它们最后的检验点重新创建调度器。因此，对于在图 3A-3D 中描绘的和在图 4 中显示的本发明的示例性应用，假设在 JVM 64 处发生崩溃事件，对于在 JVM 64 中的示例性调度器所存储的检验点信息 84' 将从本地存储器存储装置，例如硬盘驱动 94，被传送回在 JVM 64 处执行的重新启动的调度器部件。这些检验点是属于调度器的过去，因为某些信息可在采取检验点后被接收并甚至被处理。调度器因此独立地将好奇消息 85 发送到它们的原有部

分，使得原有部分重新发送丢失的消息。按照本发明，丢失的消息然后将被重新处理，并由于由本发明提供的确定性执行，将生成相同的结果。当调度器赶上它发生故障的时间时，它的状态变为与它刚发生故障之前它的状态相同。从那以后，它继续处理新的消息。

[0055]如上面提到的，对于在 JVM 中的每个调度器的检验点信息可被存储和被间歇地或递增地传送至远端机器(例如，JVM)。因此，在替换实施例中，位于远端的调度器可以通过存储来自另一个调度器的检验点而执行“被动备份”。如果被动备份被用来存储检验点，则当检测到故障时，被动备份创建备份部件的实例，即产生复制品，并变为主动，继续执行发生故障的调度器的工作，直至它重新启动并赶上丢失的计算为止。不像某些其他容错系统的主动复制那样，被动备份不执行冗余的计算。它们仅仅保持检验点状态，这样，如果工作的机器发生故障，则备份能够快速地以最小的延时接管计算。

[0056]因此，对于在图 3A-3D 中示出和现在在图 5A 中示出的本发明的说明性示例性应用，对于在 JVM 64 中的示例性调度器的软检验点信息 84 将被传送到与用于第二 JVM 64 的、被远程存储在第三 JVM 66 中的备份调度器部件 74 相关联的存储器用于在此存储；同样地，如图 5A 所示，对于在 JVM 66 中的示例性第三调度器的软检验点信息 86 将被传送到与用于第三 JVM 66 的、被远程存储在第二 JVM 64 中的备份调度器部件 76 相关联的存储器用于在此存储。因此，例如，当包括调度器 2 和备份调度器部件 76 的 JVM 64 变为禁止的或崩溃时，可生成正在 JVM 64 中执行的调度器 2 的复制品，如图 5B 所示。也就是说，在 JVM 66 中，可以根据远端被动备份产生正在 JVM 64 中执行的调度器 2 的复制品。也就是说，产生包括用于执行的第二 JVM 64 的部件的新的调度器 2'。基于最大故障检测延时和用于复制品创建的时间，可以估计在一个故障的情形下的最大断开时间，这提供了高的可用性保证。

[0057]最后，对于在图 3A-3D 中示出和现在在图 5C 中示出的本发明的说明性示例性应用，在恢复崩溃的第二 JVM 64 时，复活的调

度器 2 将获取存储在第三 JVM 66 中的所产生的复制品，即调度器 2'。因此，可以向在其他机器(例如，JVM1)上的原有部分部件告知第二 JVM 64 在工作，经由通知消息 91 运行；而且，向第三 JVM 66 告知调度器 2 的所产生的远端备份(即调度器 2')可被终止。

[0058]尽管本发明具体参照说明性的和执行的实施例被示出和描述，但本领域技术人员将会理解，可以在形式和细节上做出上述和其他改变而不背离本发明的主旨和范围，本发明的主旨和范围仅由所附权利要求的范围来限定。

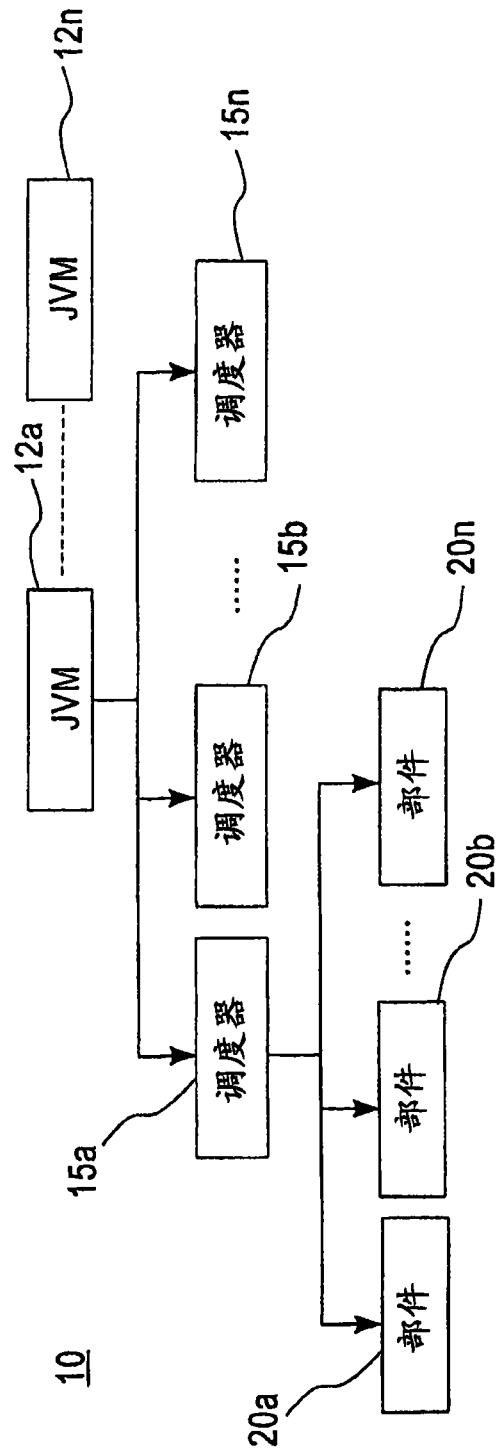


图 1

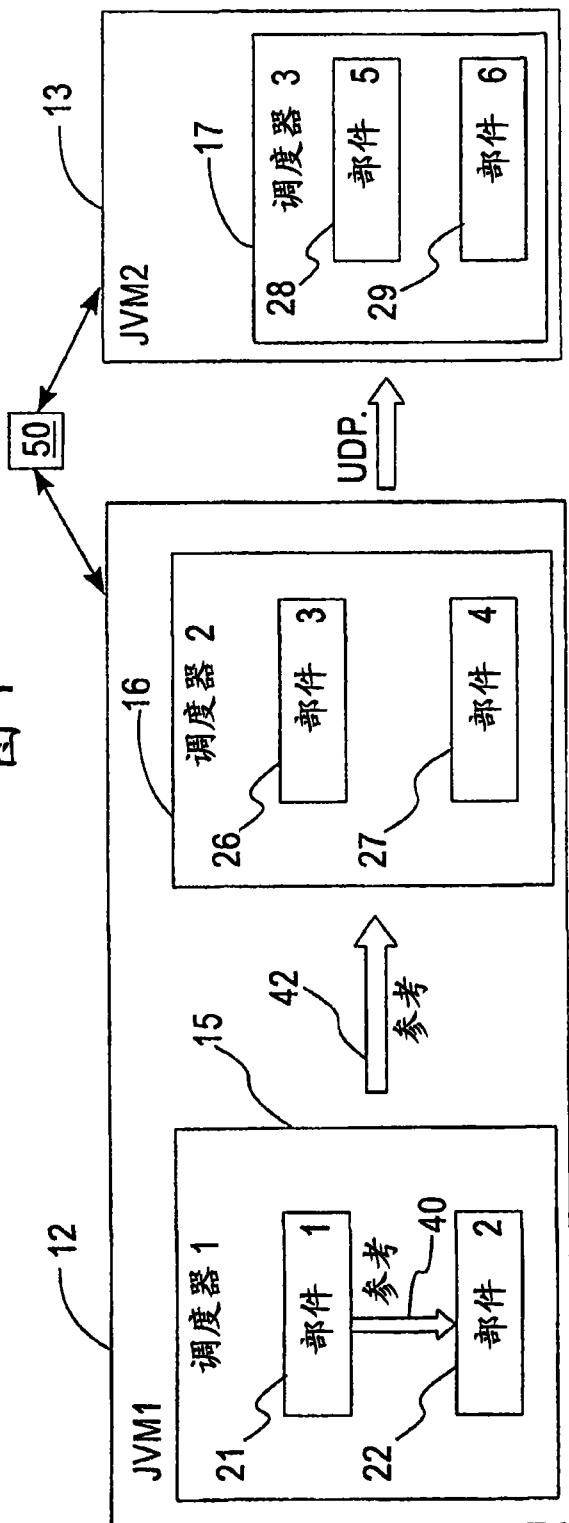


图 2

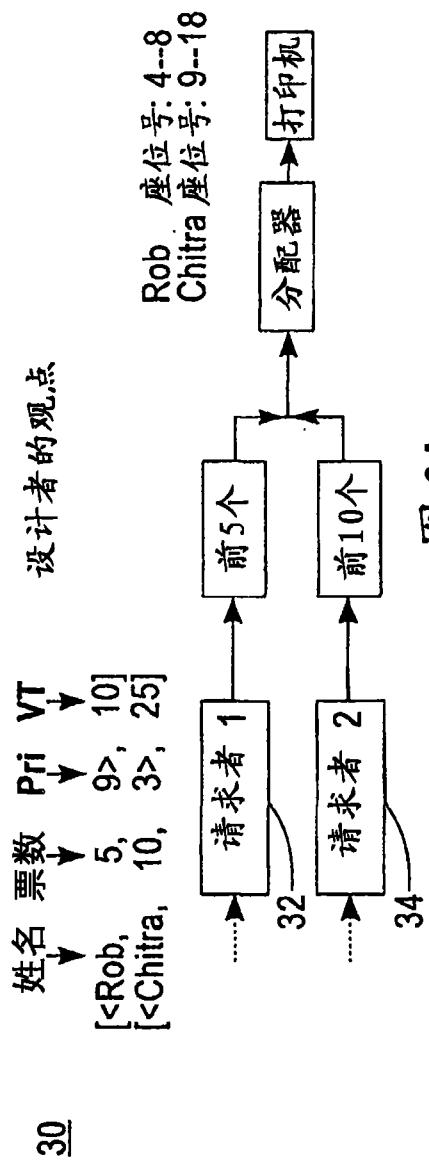


图 3A

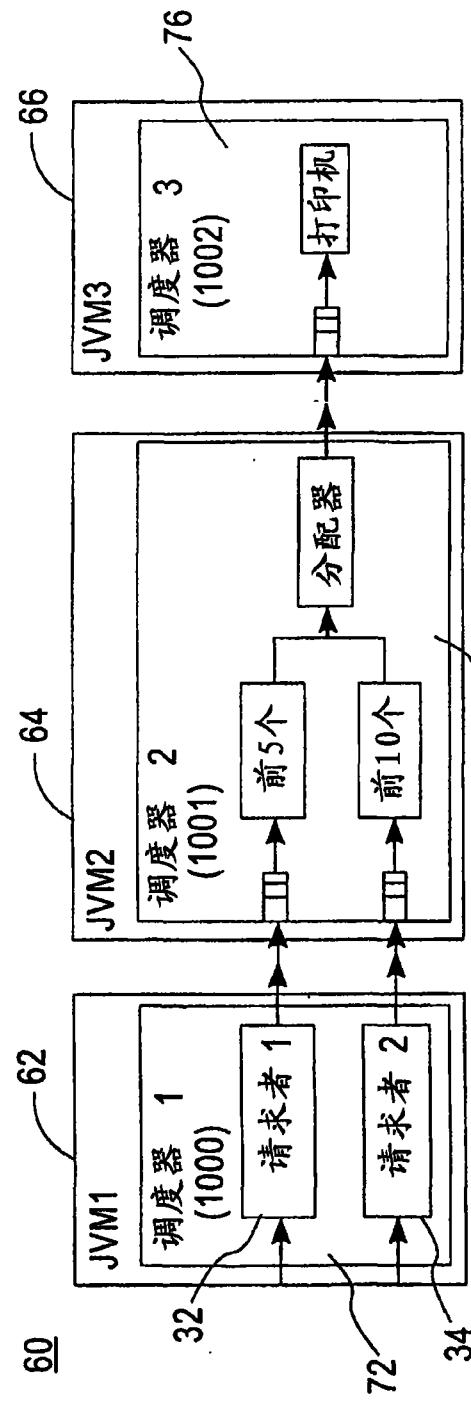


图 3B

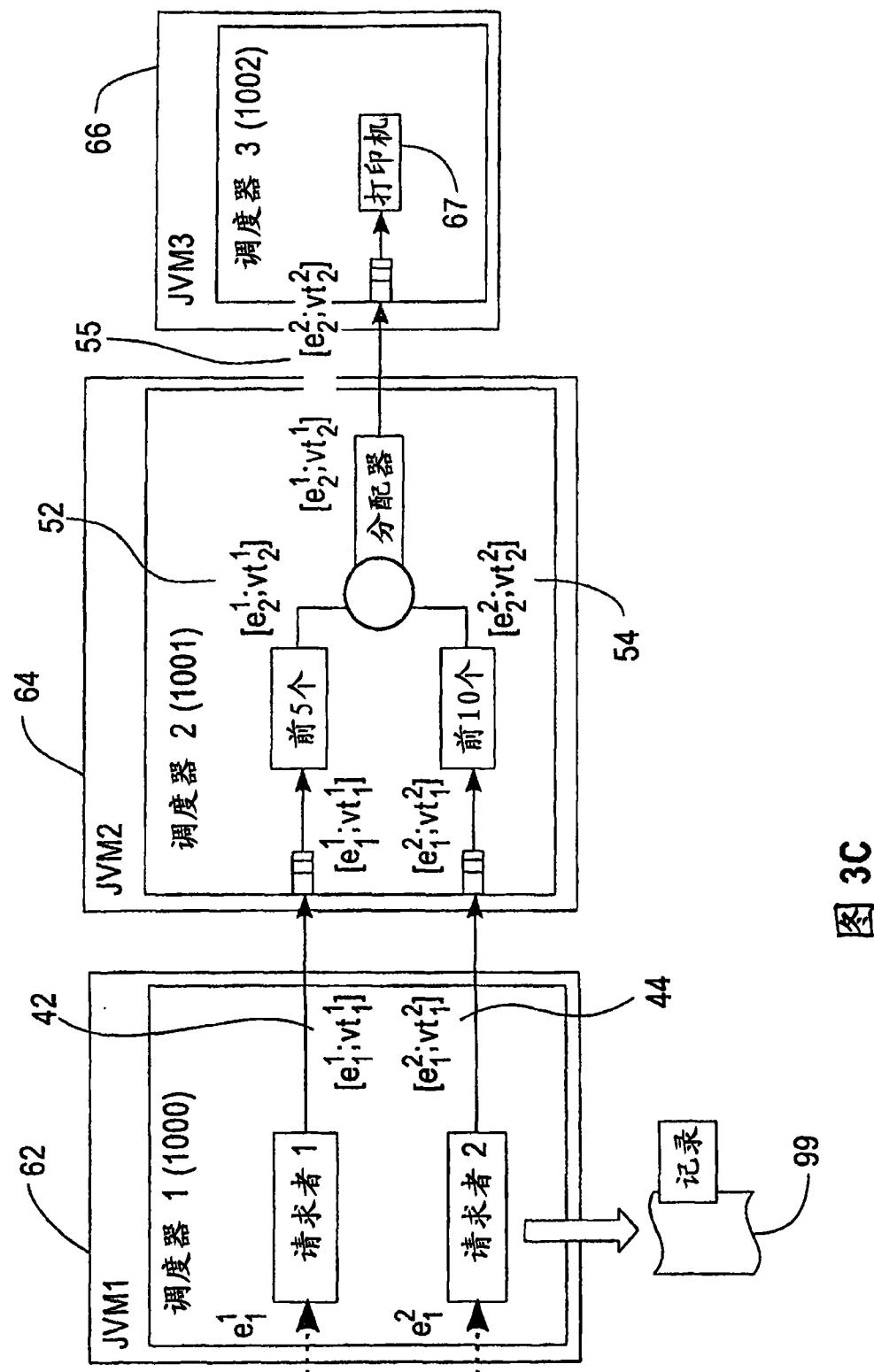


图 3C

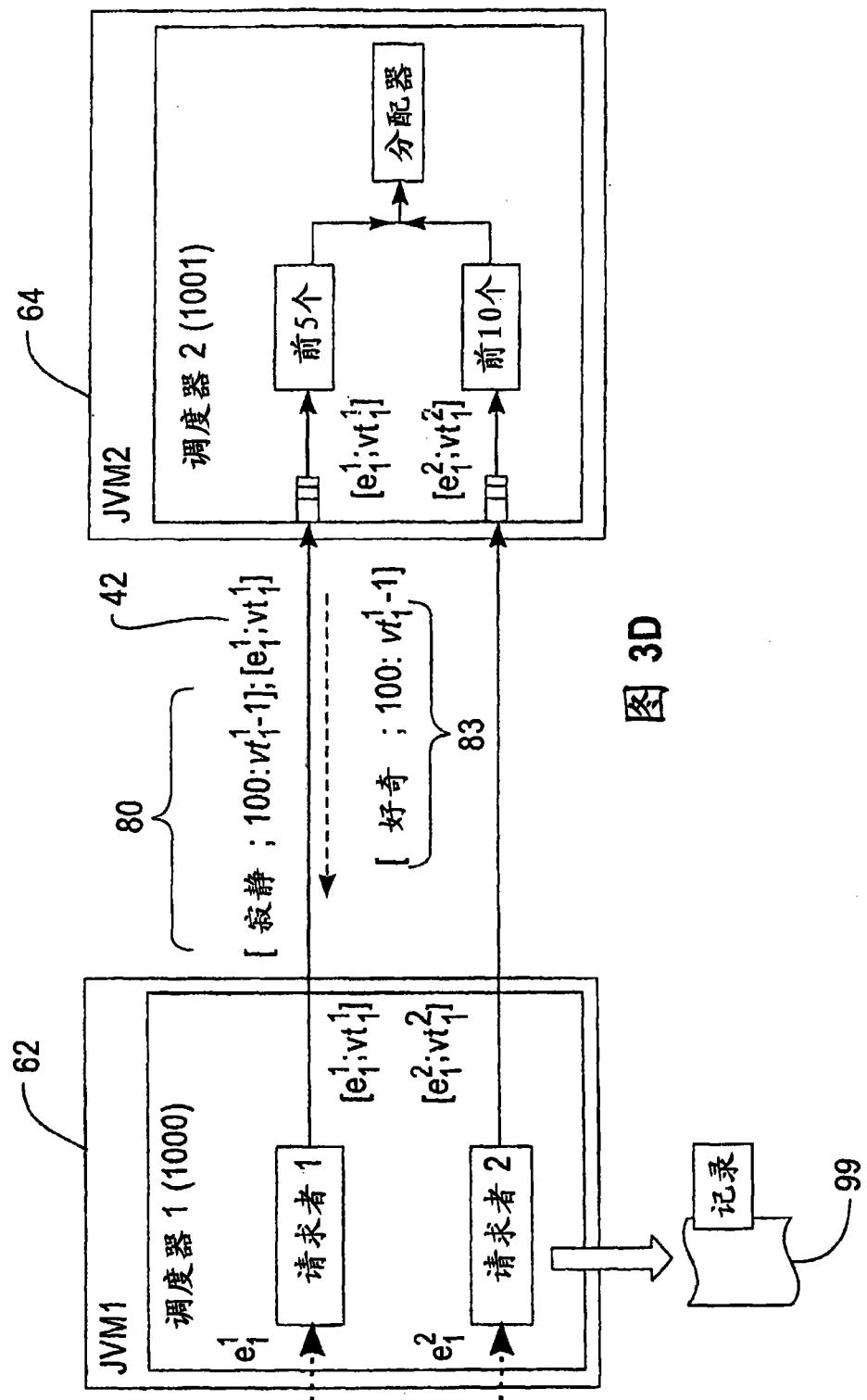


图 3D

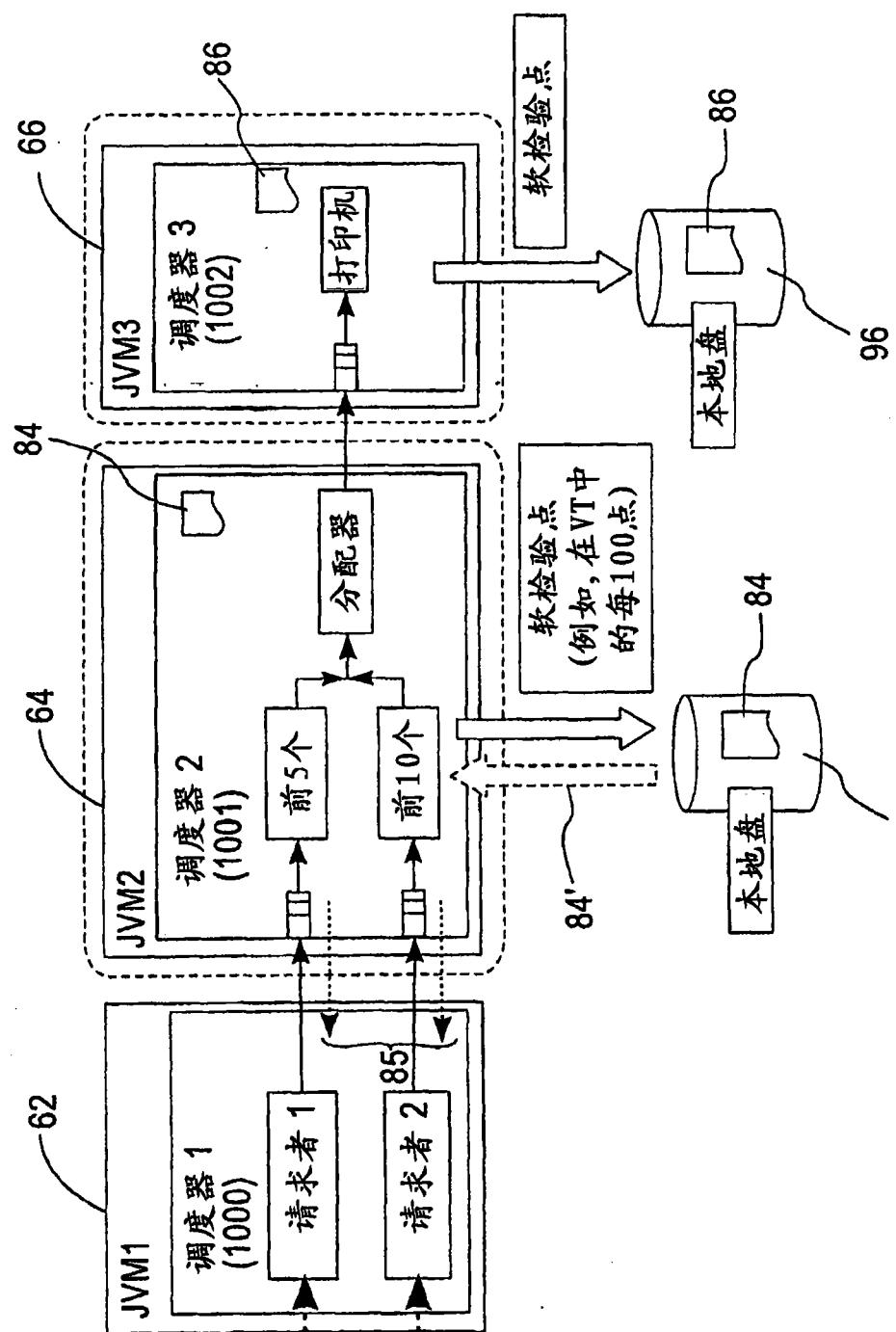


图 4

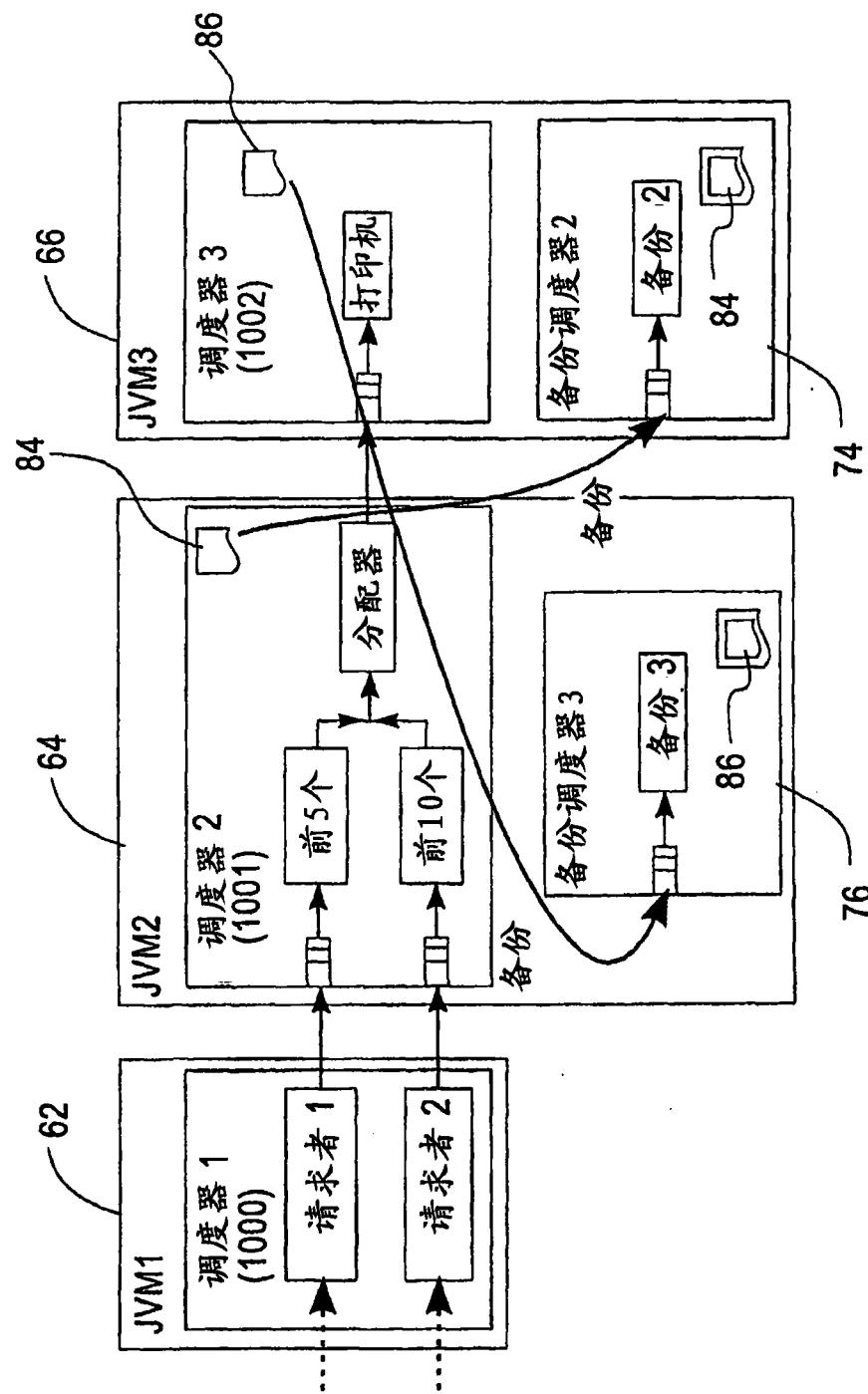


图 5A

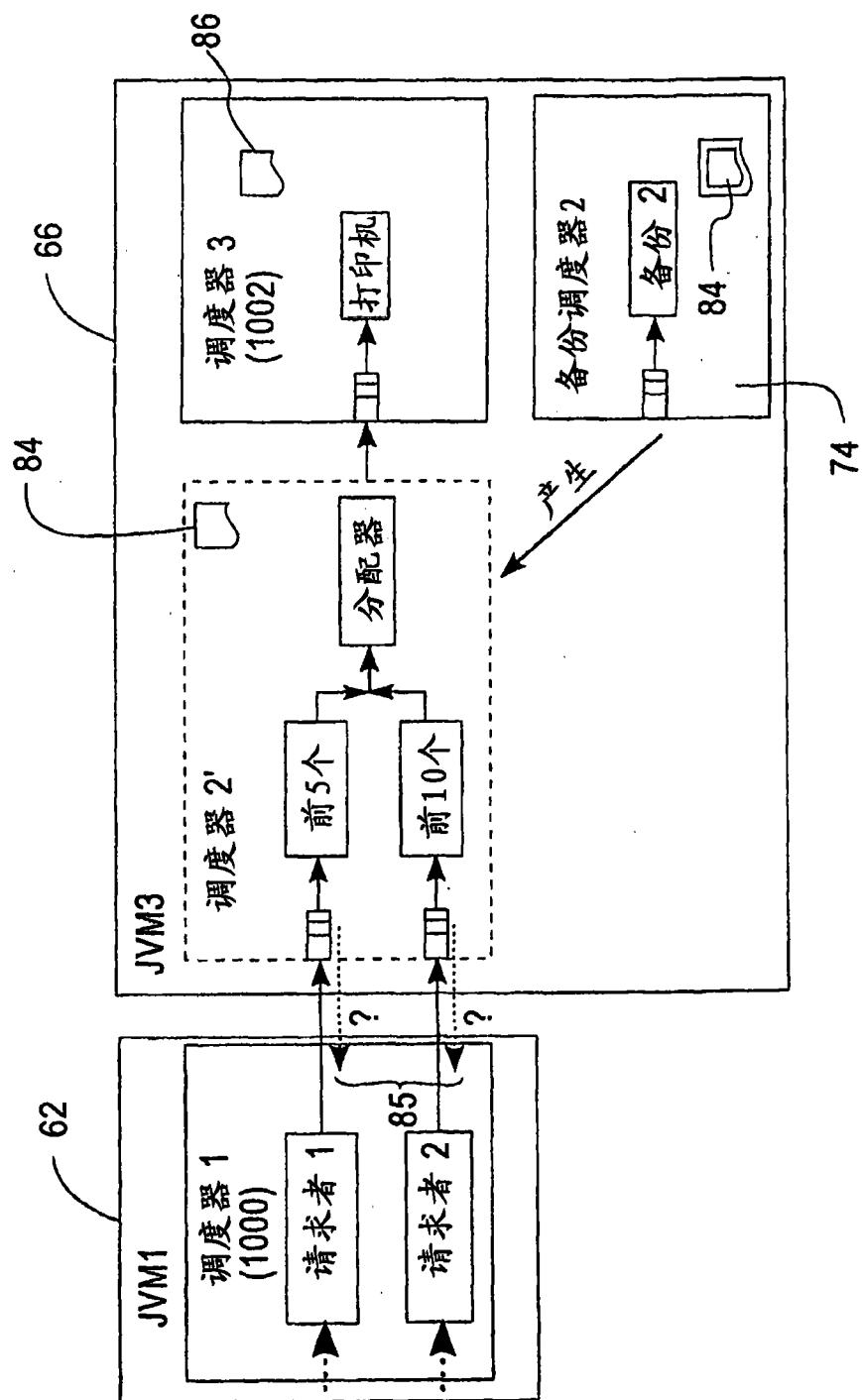


图 5B

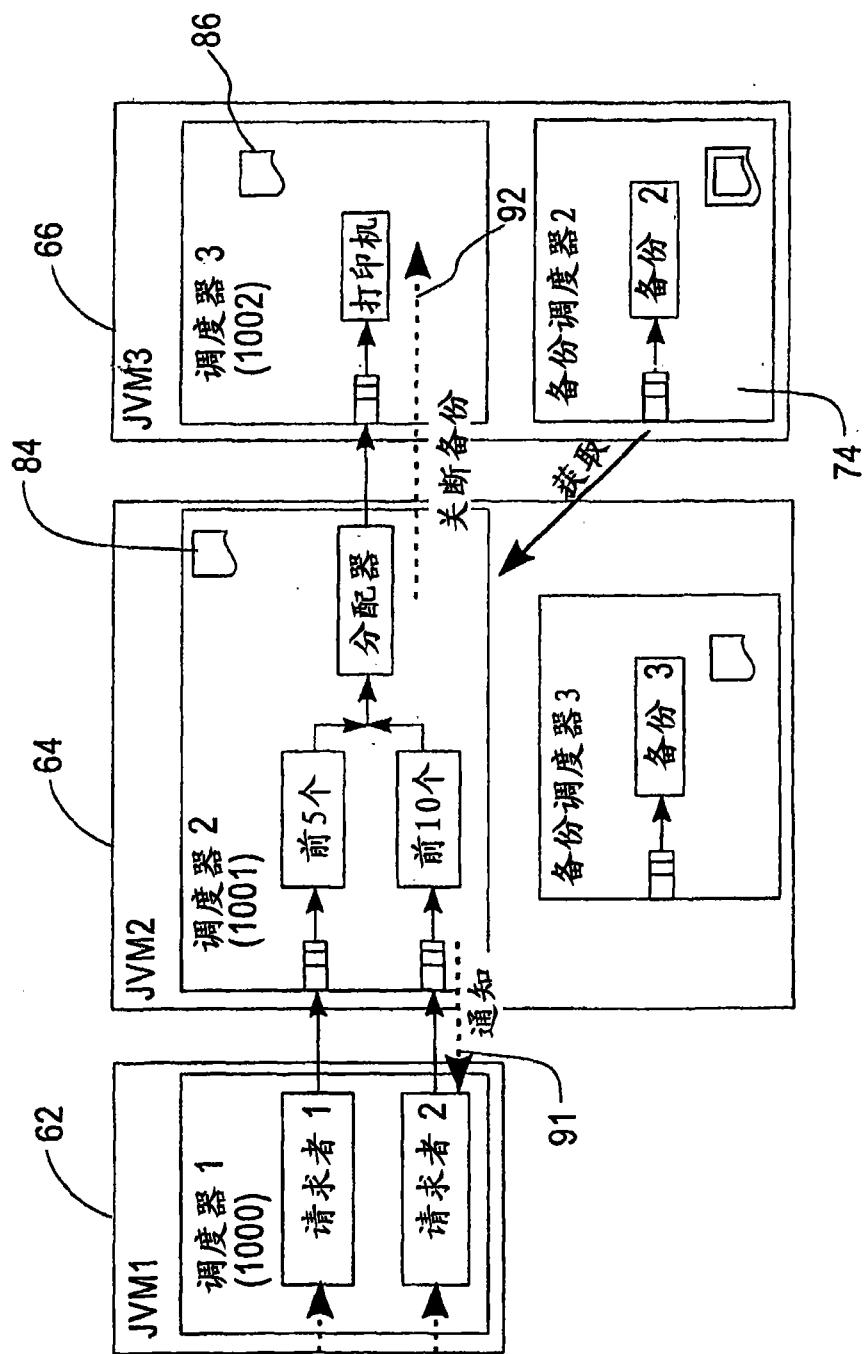


图 5C