第 20 卷 第 1 期 控 制 与 决 策 2005 年 1 月
Vol 20 No. 1 Control and Decision Jan. 2005

文章编号: 1001-0920(2005)01-0006-05

普适计算中的无缝迁移策略

张德干[1,2], 尹国成[3], 史元春[1], 徐光祐[1], 曾广平[2]
(1. 清华大学 计算机科学与技术系, 北京 100084; 2. 北京科技大学 信息工程学院, 北京 100083; 3. 东北大学 信息科学与工程学院, 辽宁 沈阳 110004)

摘 要: 针对分布式计算环境已有的一些迁移机制, 都无法满足普适计算模式下任务迁移时对无缝性的要求, 研究了任务的形式化描述, Agent 的分类以及任务的迁移粒度问题. 基于 Mobile Agent 的任务无缝迁移策略, 解决了迁移过程中的迁移算法、迁移时延、迁移失效等问题. 其特点在于基于合理的迁移粒度和更小的迁移时延, 通过 Mobile Agent 的移动保证迁移的无缝性. 多个应用场景中的测试实践表明了该策略的有效性.

关键词: 普适计算; 无缝迁移; 连续性; 迁移时延
中图分类号: TP391 **文献标识码**: A

Strategy of seamless transfer for pervasive computing

ZHANG De-gan[1,2], *YIN Guo-cheng*[3], *SHI Yuan-chun*[1], *XU Guang-you*[1], *ZENG Guang-ping*[2]
(1. Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China; 2. School of Information Technolgy, Beijing University of Science and Technology, Beijing 100083; 3. School of Information Science and Technology, Northeastern University, Shenyang 110004, China. Correspondent: ZHANG De-gan, E-mail: zdg@neuera.com)

Abstract: To the seamlessness of task transfer of pervasive computing paradigm. The formal description of task is discussed. Then, classification of agent and transferring granularity of task are suggested. The efficient strategy of seamless transfer of task is presented and designed, including the problem seamless transferring method, transferring delay, transferring failure. The key insight is that seamlessness is based on mobile agent under adaptive transferring granularity of task and less transferring delay. Scenarios of implemented demo show the efficiency of the strategy.

Key words: pervasive computing; seamless mobility; continuity; delay of transfer

1 引 言

普适计算(Pervasive/Ubiquitous Computing)是计算、通信和数字技术等多种技术的融合, 它把信息空间与人们生活的物理空间集成在一起成为一个整体, 从而使计算和通信象水、电、空气一样成为生活必需品. 普适计算的研究促进了计算与通信相结合, 方便了用户的信息访问. 但要使计算和通信真正无所不在地进入人们生活, 从而进一步提高工作效率和生活质量, 还需要研究无缝移动(Seamless Mobility)等多方面的重要课题[1~4]. 无缝移动是指用户或用户的任务在移动过程中, 与该任务相关的历史状态、上下文信息也同时随着移动, 而用户周围可用的计算环境和软件资源也动态地发生着适应性的变化. 无缝移动的功能需求主要体现在连续性上. 连续性指无缝移动可以暂停, 也可以继续, 但不能丢失程序的状态信息和历史信息. 迄今为止, 无缝移动研究的先行探索者中最具影响的两个计划是美国 MIT 的 Oxygen 研究计划和卡耐基梅隆大学(CMU)的 Aura 研究计划.

尽管国内外有些研究者针对分布式计算环境提

收稿日期: 2004-03-01; **修回日期**: 2004-04-06.
基金项目: 国家自然科学基金项目(60103004).
作者简介: 张德干(1970—), 男, 湖北黄冈人, 博士, 从事信息融合、普适计算技术的研究; 徐光祐(1940—), 男, 上海人, 教授, 博士生导师, 从事计算机视觉、普适计算技术的研究.

出了一些迁移机制[5~7], 但都无法满足普适计算模式下的无缝性要求. 鉴于此, 本文研究了任务的无缝迁移策略, 包括对迁移时延、迁移失效等问题的讨论. 显然, 它与平滑越区切换、无缝漫游不是同一层次上的概念.

2　任务的形式化描述

由于任务的多样性, 构成任务的子任务也存在着很大的差别, 对于不同的子任务, 完成的方式也不尽相同. 根据子任务所包含数据类型的性质差异, 主要分为以下 3 种形式: 事件型(Event), 流媒体型(Stream), 批量型(Bulk). 一般地, 一个任务由上述 3 种类型的子任务组合而成. 当任务只涉及上述其中的某一种执行方式时, 可用 XML 进行形式化描述. 当任务包含上述 3 种执行方式且执行顺序与时间有关时, 可用 SMIL[8]进行形式化描述. 例如:

```
〈smil xmlns = " http: //www. w3. org/2001/SMIL20/Language"〉
  〈head〉〈layout〉
    〈root- layout width= "…" height= "…"/〉
    〈region id= "…" left= "…" top= "…" width = "…" height= "…"/〉
  〈/layout〉〈/head〉
  〈body〉
   〈seq〉
    〈par〉
     〈img src= "…" region= "…" id= "…"/〉
     〈excl dur= "indefinite"〉
       〈video id= "…" src= "…" region= "…" begin= "…activateEvent"/〉
     〈/excl〉
    〈/par〉
   〈/seq〉
  〈/body〉
〈/smil〉
```

其中: 〈seq〉…〈/seq〉表示子任务序列执行时间拓扑关系, 而〈par〉…〈/par〉表示可以并行执行的子任务序列.

3　MA &MAS

基于 Mobile Agent(MA)的计算是普适计算模式中的一种重要途径[4~8]. 主动、智能化的 MA 一方面充当了应用组件的运行容器, 另一方面也可以根据应用特征和当前网络状况进行有目的的动态资源部署或调整. Agent 作为一种特殊的计算资源, 所产生的直接效果之一是支持系统计算资源的动态部署和自由移动. 从性能角度, 基于这种方式的系统是很容易调整和管理的, 典型的行为如时延控制、负载调节、无缝迁移等, 因而适合无缝移动的应用.

MA 是指能够在异构网络 N 中自主地从一个网络节点 PA_i 迁移到另一个网络节点 PA_j $(i \neq j)$, 并与其环境进行交互的有名字的智能软件程序实体. 该程序能自主地决定在什么时候迁移到什么地方. 它能在程序运行的任意点上挂起, 然后迁移到另一台主机上, 并接着这一点继续往下执行. MA 也能克隆自己或产生子 MA, 迁移到其他主机上来共同协作完成复杂的任务. 除具有一般 Agent 的属性即自治性、主动性、智能性外, 还具有移动性、自适应性, 可以迁移到资源集中的网络节点或在多个网络节点之间移动、漫游.

MAS (Mobile Agent System) 是指能 CreateMA (创建 MA), SchduleMA (调度 MA), ExecuteMA (执行 MA), TransferMA (迁移 MA) 和 DestroyMA (撤销 MA) 的管理平台, 它由 MAS_Name (名称) 和 MAS_Address (地址) 唯一标识. 每个 MAS 都可以运行多个 Mobile Agent. MA 通过与主机服务进行交互来获得所需的服务. 基于 MA 技术构建的系统, MA 之间可通过 MA 通信原语进行交互, 基于共享语汇进行协商, 多个 MA 通过协作完成任务[7,8].

4　Agent 的分类与任务的迁移粒度

多 Agent 系统中包含一套 Agent, 按照功能可分为多种, 这里给出以下 3 种: 1) 用户终端代理 TA (含用户代理 UA); 2) 导航代理 VA (网络能力代理 NCA, 位置管理代理 LMA); 3) 任务代理 WA (执行代理 EA —— 提供者代理 RA 和代码运行代理 CEA, 数据代理 DA —— 服务代理 SA 和用户文档数据库代理 UDDA), 这些 Agent 有的是固定的, 有的是 MA.

根据所迁移的内容完整与否, 任务的迁移粒度可分为强迁移(全部迁移)和弱迁移(部分迁移)两种类型, 迁移方式可以有随机命令和旅行计划两种. 强迁移要求在异构平台之间传输与任务有关的所有内容(代码、数据、执行状态), 到达目标站点后, 能够从移动前的断点处恢复后继续往下执行. 但因为在异构平台下收集和恢复执行状态数据困难较大, 需要有执行状态的中间表示格式, 同时会消耗较大的网络带宽, 因此这种迁移的实现大大增加了系统的负担和复杂性. 弱迁移只迁移部分执行状态或数据, 其速度较强迁移快, 但不能保存 MA 的完整运行状态.

5　基于 Agent 的无缝迁移策略

迁移一个任务, Agent 一般要完成的基本操作步骤如下: 决定迁移在原站点上正在运行的某个

Agent; 将该 Agent 挂起; 重建该 Agent 所需的信息, 传送到目的站点; 在目标站点恢复运行该 Agent. 于是, 任务迁移策略的设计要从以下两个角度考虑: 1) 从原站点传送到目标站点的任务是全部信息还是部分信息; 2) 从原站点上挂起 Agent 和在目标站点上恢复运行 Agent 的时机角度.

5.1 迁移算法

根据上述对 Agent 的分类, 本文作如下规定:

1) 导航 Agent(VA)无需完成与具体应用直接相关的任务, 它为特定的网络寻址而设计, 熟悉目标子网的拓扑结构. 导航 Agent 的数据结构分为两部分: 一部分是导航 Agent 自身的"功能体", 另一部分是用于装载被移动对象的"信箱(MB)", 通过"信箱"可有效地运送一个或多个任务 Agent 在子网中迁移.

2) 任务 Agent(WA)实现具体的应用功能, 它在每个被访问节点上提供执行代码、数据以及状态环境管理等功能, 它可以在子网之间跟随导航 Agent 迁移, 而无需熟悉它所访问的子网结构.

3) 迁移时, 任务 Agent 先找对应的导航 Agent, 并进入其"信箱"中, 随后由导航 Agent 将来访的任务 Agent 运送到要访问的目标站点上去.

为方便起见, 作如下假设:

假设 1 某 TA 希望逻辑节点 PA_2 上的 WA(这里不详细区分是执行代理 EA, 还是数据代理 DA)迁移至另一逻辑点 PA_3, 根据迁移对象的时间拓扑关系已制定好了迁移旅行计划(一种独立于 Agent 本身的数据结构). 当前场景是: 该 TA 正与逻辑节点 PA_1 相连, 而逻辑节点 PA_1 通过双向逻辑链路与逻辑节点 PA_2 相连, 用实线箭头表示 WA 能够横越的逻辑链路方向.

本文设计的 WA 迁移的具体算法如下:

1) 按照事先制定好的迁移旅行计划, 逻辑节点 PA_2 利用 VA 按逻辑节点 PA_3 提供的目标站点地址以某种寻址方式在移动互联网中建立握手或请求连接信息, 当确认连接建立成功后, 由 VA 发送 TransferNode 消息至目标逻辑节点 PA_3, 通知其开始迁移任务. VA + WA 经打包后分组传送到目标逻辑节点 PA_3. 该包由已经记录保存的 WA 的结构, WA 相互间的关联关系, WA 占用的空间大小, WA 的类型和驻留在此逻辑节点上与任务移动有关的 VA 信息以及调度所有 Agent (导航 Agent, 执行 Agent 和数据 Agent 等)的 Messenger 信息等构成到达目标逻辑节点 PA_3 的 Messenger 不是立刻处于"执行"工作状态, 而是处于"等待"状态存储在逻辑节点 PA_3 的队列中.

2) 目标逻辑节点 PA_3 发送 UpdateLinking 消息给所有与逻辑节点 PA_2 相连的逻辑节点, 这里如 PA_1. 此消息包含更新相连链路地址信息所需要的信息, 象链路 ID, 链路两端的 IP 和端口号 Port 等. 在不断地迁移过程中, 所有到达目标逻辑节点 PA_3 上的 Messenger 存储在对应的队列中而不被执行, 直到整个迁移工作完成为止.

3) 当与逻辑节点 PA_2 相连的逻辑节点 PA_1 收到 UpdateLinking 消息后, 它关联到新链接的链路, 并发送 LinksUpdated 消息到逻辑节点 PA_2.

4) 当逻辑节点 PA_2 收到所有期望收到的 LinkUpdated 消息后, 发送 ActiveNode 消息至目标逻辑节点 PA_3. 此消息包含第 1 步之后到达的所有 Messenger 的列表, 同时逻辑节点 PA_2 删除已迁移的 VA + WA.

5) 当目标逻辑节点 PA_3 收到 ActiveNode 消息后, 该节点将把从逻辑节点 PA_2 接收到的 Messenger 的列表附在它自身的队列中. 根据该列表上的拓扑关系, 遵循 FIFO 的原则, 开始激活目标逻辑节点 PA_3 上的所有 Messenger, 至此迁移完成.

6) 当所有 Messenger 被激活后, 各 WA 开始恢复运行环境, 按顺序执行任务 ExecuteTask.

7) 在各 WA 执行过程中, 它一方面记录保护现场(WA 的结构, WA 相互间的关联关系, WA 占用的空间大小, WA 的类型以及调度所有 Agent 的 Messenger 等); 另一方面, VA 不间断地监听 ListenTask 来自外部的(这里如 PA_1)下一次将要迁移的指令 TransferSignal.

8) 在 WA 执行任务过程中, 若 VA 没有接收到继续迁移的指令(即 TransferSignal 一直为 0x0), 则 WA 将连续执行任务, 直到任务完成为止; 若 VA 接收到了继续迁移的指令(即 TransferSignal = 0x01), 则停止执行 StopTask, Goto 1 开始准备迁移, 其过程是"无缝"的.

5.2 迁移失效问题的解决策略

在普适计算环境中, Agent 的位置可能经常变动, 因此经常会有如下情况发生: Agent1 向站点 C 上的 Agent2 迁移, 并嵌入其中协同完成任务. 但在迁移过程中, Agent2 则从站点 C 迁移到了站点 D, 因而当 Agent1 到达站点 C 时, 已经无法找到 Agent2 了. 人们把这种在迁移过程中目标 Agent 发生站点位置变化, 从而导致被迁移对象不能到达目标 Agent 的现象称为迁移失效. 该问题使得任务的完成不能保持连续性, 即无缝性被打破. 解决该问题, 需要考虑以下 3 个因素: 1) 当 Agent 位置改变时, 其他 Agent 如何知道这一变化; 2) 当 Agent 迁

移时, 如何处理正发送给它的消息; 3)Agent 迁移过程中, 接收 Agent 是否可以自由迁移. 具体解决方法如下:

1) 当 Agent 位置改变时, 以发送消息的形式通知所有其他 Agent.

2) 每个 Agent 迁移前应先查询接收 Agent 的当前位置, 并将迁移关系和迁移事件以消息形式通知接收 Agent.

3) 确定 Agent 的迁移拓扑关系, 按照 FIFO 原则, 在每个 Agent 开始迁移的同时锁定与之对应的有迁移计划的但时间稍晚的接收 Agent; 迁移结束后, 解除接收 Agent 的锁定. 为此, 可引入一个信号量 semaphore 进行标识.

消息的通知形式可根据应用的实际需要采用单播(Unicast)、组播(Multicast)以及广播(Broadcast)等, 具体类型细分为: 1)GROUP 内的单播、组播、广播; 2)不同 GROUP 间的单播、组播、广播; 3)单个 Agent 向 MAS 的广播; 4)不同 MAS 间的单播、组播、广播. Agent 间消息通知形式确定为: Agent_Message_Notify (Sender, Receiver, Message).

相应地, 当 Agent 进行单播、组播和广播时, 消息通知形式分别为:

Agent_Message_Notify (Agent1, Agent2, Message);

Agent_Message_Notify (Agent, Multicast (AgentX), Message), AgentX 为其他的 Agent;

Agent_Message_Notify (Agent, Broadcast (Any), Message), Any 为 Group 内外任意的 Agent.

基于时间迁移拓扑关系而设计的"先寻址再锁定发送"的同步通信机制, 能够在迁移 Agent 和目标 Agent 上实现迁移与通信的同步, 可从根本上避免迁移失效现象的发生, 而且具备对各种应用模式的适应能力, 在避免失效的同时, 减少了对目标 Agent 迁移自由的限制, 因而更具通用性. 其中迁移的时间拓扑关系可反映在迁移旅行计划中. 旅行计划由若干旅行序列构成, 每个旅行序列包含的数据结构为:

{

计划标识 TP_ID, 计划名称 TP_Name, 计划制定时间 TP_Time, 旅行序列号 No, 迁移对象 TP_Object, 迁移粒度 TP_Granule, 源地址 TO_IPC, 目标地址 TO_IPD, 迁移条件 TP_Condi, 迁移标记点 TP_SnapshotPoint, 恢复运行的入口地址 TP_RunEntry

}

数据结构中的"迁移条件 TP_Condi, 迁移标记点 TP_SnapshotPoint 和恢复运行的入口地址 TP_RunEntry"是迁移行为的实质性动作, 即当满足"迁移条件 TP_Condi"时, 旅行计划才开始启动. 同时, 记录并保存"迁移标记点 TP_SnapshotPoint"和"恢复运行的入口地址 TP_RunEntry", 这两者为迁移对象在"目标地址 TO_IPD"处恢复运行保留当前的现场. "迁移条件 TP_Condi"的设定以及是否满足主要考虑以下几方面:

1) Agent 的当前状态(本文把 Agent 分为 5 种状态: 就绪态 1, 等待态 2, 迁移态 3, 运行态 4 和死亡态 5, 它们分别对应不同的任务需求)是否为迁移等待态;

2) 目标地址 TO_IPD 是否可达;

3) 无缝性时延阈值是否满足和迁移过程中的残余依赖现象评价等.

5.3　迁移时延的降低策略

一个 Agent 从在初始运行的原站点上被中止到迁移至目标站点上恢复运行的这段时间间隔称为 Agent 的迁移时延. 迁移时延是移动无缝性评价的主要参量之一. 迁移一个任务涉及到的信息有: 指令空间、地址空间、挂起时刻的运行状态、运行代码、数据和调度控制信息(Messenger)等. Messenger 所包含的调度控制信息和挂起时刻的运行状态必须全部迁移, 而其他信息根据需要不必全部迁移. 一旦在目的站点重建了足够的必要信息, 尤其是恢复了迁移前的运行状态的 Snapshot 后, 该 Agent 便可以重新开始运行.

经分析发现, 迁移时延的长短与以下 3 个因素有关:

1) 迁移粒度, 即强迁移还是弱迁移. 传送部分信息时, 应选择哪一部分信息先传送是关键, 一般说, 应先传送在目标站点上恢复运行所必需的信息, 即核信息集. 例如运行代码和挂起前捕获的运行状态等.

2) 何时迁移一个 Agent, 即 Agent 挂起时间. "挂起"机制分为决定迁移后立即挂起, 当全部信息传送完毕后挂起, 当核信息集传送完毕后挂起 3 种情况.

3) 何时恢复 Agent 开始运行, 即 Agent 恢复时间. "恢复"机制分为全部信息传送完毕后恢复和核信息集传送完毕后恢复 2 种情况.

下面在基本迁移策略的基础上讨论无缝移动实现的新机制问题. 实际执行中, 上述 3 种因素可组成有效的迁移方式有以下 3 种:

1) 强迁移, 决定迁移后挂起, 完成全部的传送

后恢复;

2) 强迁移, 决定迁移后挂起, 完成核信息集的传送后恢复, 同时继续传送剩余的全部信息.

3) 弱迁移, 决定迁移后挂起, 完成核信息集的传送后恢复, 同时有选择地传送其他部分信息.

第 1 种方式的迁移时延包括全部信息打包并传送, 目标站点上恢复重建该 Agent 和全部信息; 第 2 种方式的迁移时延包括核信息集打包并传送, 目标站点上恢复重建该 Agent 和核信息集(在目标站点运行该 Agent 的同时, 原站点继续传送剩余信息); 第 3 种方式的迁移时延包括核信息集打包并传送, 目标站点上恢复重建该 Agent 和核信息集, 在目标站点运行该 Agent 的同时, 根据需要有选择地传送其他部分信息到达目标站点.

在相同的评价体系下, 三者相比较, 显然第 1 种方式的迁移时延最长, 第 3 种方式的迁移时延最短. 因此, 采用第 3 种方式较为适宜.

6 仿真测试

利用上述无缝迁移方法, 可实现任务在位于 House, Office, Coffee house, Park, Stadium, Airport 等不同场景下的处于固定状态的 PC, 可携带任意移动的 PDA, 笔记本电脑 laptop 等设备群之间无缝迁移[13]. 下面给出一个被迁移的示例任务描述:

```
〈smil〉〈head〉〈layout〉
    〈root-layout background-color = "#D3DD86" width= "640" height= "480" /〉
    〈region id= "videoregion" top= "0" left= "0" width= "320" height= "240" /〉
    〈region id= "textregion" top= "0" left= "321" width= "320" height= "159" -〉
  〈/layout〉〈/head〉〈body〉〈seq〉〈par〉
    〈video src = "SmartMeetingRoom.avi" region= "videoregion" begin= "0.6s" /〉
    〈audio src= "ActiveSpace.mp3" begin= "2.6s"/〉
    〈textstream src = "MeetingRoom.txt" region= "textregion" begin= "5s" end= "9000s" /〉
  〈/par〉
 〈/seq〉
〈/body〉
〈/smil〉
```

在该示例任务中, 有 3 个具有时序拓扑关系的可并行执行的子任务: 播放视频文件 SmartMeetingRoom.avi; 播放音频文件 ActiveSpace.mp3; 编辑文档文件 MeetingRoom.txt. 随着人从某一个节点(例如 House)移动到另一个新的节点(例如 Airport), 利用本文所研究的任务无缝迁移功能, 可将 House 中 PC 机上未看完的视频文件 SmartMeetingRoom.avi, 未听完的歌曲文件 ActiveSpace.mp3 和未写完的文档文件 MeetingRoom.txt 迁移到 Airport 候机大厅随身携带的笔记本或 PDA 上续前继续看完、听完和写完.

7 结 语

本文以普适计算中无缝移动所关注的"连续性"为主要实现目标, 研究了任务的描述, 任务的类型划分, Agent 的分类与任务的迁移粒度, 以及基于 Agent 的任务无缝迁移策略; 解决了任务迁移过程中所涉及的迁移算法、迁移时延、迁移失效等问题. 多个应用场景的测试表明, 本文方法是有效可行的.

面向普适计算的无缝移动技术的研究是一个新课题, 还存在很多有待进一步研究的问题. 例如支持连续性的上下文获取与存储, 支持自适应的资源动态配置, 以及移动环境的不确定性、多代理的交互与协作和移动代理的安全等.

参考文献(References)

[1] Simmons R, Apfelbaum D. A task description language for robot control[A]. *Proc Conf on Intelligent Robotics and Systems*[C]. New York, 2001: 138-147.

[2] Garlan D, Siewiorek D P. *Project aura: Toward distraction-free pervasive computing* [M]. New York: IEEE Pervasive Computing, 2002.

[3] Ciancarini P. Coordinating multi-agent applications on the WWW: A reference architecture[J]. *IEEE Trans on Software Engineering*, 2002, 24(5): 363-375.

[4] Satyanarayanan M. Pervasive computing: Vision and challenges [J]. *IEEE Personal Communications, August*, 2001, 5(8): 10-17.

[5] David Kotz, Robert S Gray. Mobile agents and the future of the internet[J]. *ACM Operating Systems Review*, 2002, 33(3): 7-13.

[6] Karnik N M. Design issues in mobile agent programming systems[J]. *IEEE Concurrency*, 1999, 6(3): 125-132.

[7] Danny B Lange, Mitsuru Oshima. Seven good reasons for mobile agents[J]. *Communications of the ACM*, 2001, 42(3): 86-89.

[8] Chen E Y, Zhang D G, Shi Y C, et al. A programming framework for service association in ubiquitous computing environments[A]. *IEEE PCM*[C]. Singapore, 2003: 136-142.