[19] 中华人民共和国国家知识产权局

[51] Int. Cl[7]
G06F 9/06
G06F 9/455 G06F 9/45

[12] 发明专利申请公开说明书

[21] 申请号 03107311.5

[43] 公开日 2003 年 10 月 8 日

[11] 公开号 CN 1447224A

[22] 申请日 2003.3.20 [21] 申请号 03107311.5

[30] 优先权

[32] 2002. 3.21 [33] US [31] 10/104751

[71] 申请人 惠普公司

地址 美国加利福尼亚州

[72] 发明人 J·A·科哈 A·卡卡雷

T·C·奥康斯基

[74] 专利代理机构 中国专利代理(香港)有限公司

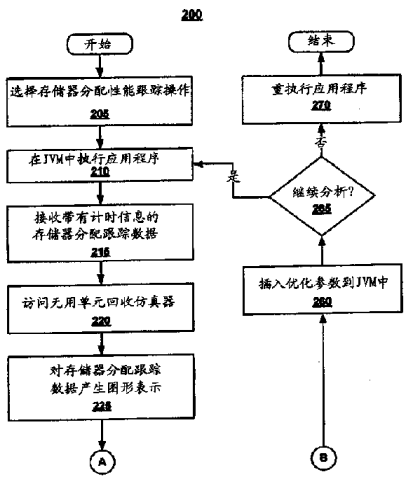
代理人 吴立明 王 勇

权利要求书 1 页 说明书 10 页 附图 8 页

[54] 发明名称 计算机应用程序中优化存储器使用的方法

[57] 摘要

优化计算机应用程序中存储器使用的方法。 接收(215)存储器使用数据(720)，其中存储器使用数据(720)包括计时信息。 产生(225)存储器使用数据(720)的图形表示(740)。 接收(230)至少一个堆栈参数(725)。 根据存储器使用数据(720)和堆栈参数(725)，运行(245)存储器使用仿真。



ISSN1008-4274

1. 优化计算机应用程序（710）存储器使用的方法，该方法包括:

a) 接收（215）应用程序存储器使用数据（720），该应用程序存储器使用数据（720）包括计时信息;

b) 产生（225）该应用程序存储器使用数据的图形表示（740）;

c) 接收（230）至少一个堆栈参数（725）; 和

d) 根据该存储器使用数据（720）和该堆栈参数（725）操作（245）存储器使用仿真。

2. 权利要求 1 中的方法，其中计算机程序是基于 Java 的。

3. 权利要求 1 中的方法，还包括插入（260）堆栈参数到程序语言解释器。

4. 权利要求 3 中的方法，其中程序语言解释器是 Java 虚拟机。

5. 权利要求 1 中的方法，其中堆栈参数（725）响应用户输入而被接收。

6. 权利要求 1 中的方法，还包括重复 c)和 d)优化无用单元回收。

7. 权利要求 1 中的 方法，还包括产生（250）无用单元回收仿真的图形表示（740）。

8. 权利要求 1 中的方法，其中存储器使用数据（720）由存储器分配性能跟踪接收。

9. 权利要求 8 中的方法，其中存储器分配跟踪作为-Xverbosegc 数据记录。

10. 权利要求 1 中的方法，其中 d)包括:

接收（605）存储器使用数据;

根据存储器使用数据确定（610）仿真模型;

根据存储器使用数据确定（615）堆栈配置和堆栈策略;

接收（620）至少一个堆栈参数;

为存储器使用数据的至少一个测量间隔确定（625）仿真特性;

根据仿真特性仿真（630）一个应用程序运行。

计算机应用程序中优化存储器使用的方法

技术领域

本发明的各种实施方案涉及到计算机编程领域。

背景技术

许多现代的编程语言提供了为应用程序实时创建的对象或其它数据结构指令实施动态存储器分配的可能性。一些语言，如 C 和 C++明确要求用户执行诸如存储器分配的回收，而其它语言，如 Java 和 C# 提供对存储器管理的支持，允许为动态存储器分配提供动态回收。例如：Java 程序语言在 Java 虚拟机（JVM）堆栈中分配对象。当对象不再需要时，它不明确地在堆栈中被释放或迁移。为了迁移不使用的对象，Java 提供了内嵌的回收不被引用的动态分配对象功能，称为“无用（存储）单元”。动态存储器管理能力通常被称作“无用单元回收”。

当存储器不再被需要时，无用单元回收提供了释放和回收存储器的功能，提供了一个能够创建更有效应用程序的范例。然而，无用单元回收本身在许多 Java 应用程序中增加了操作开销。特别是，使用重要的系统存储器资源的 Java 企业应用程序能够影响到它们运行增加作为工作负荷的无用单元回收活动。这就导致 Java 企业应用程序的运行缓慢和某些情况下不可靠的操作。

除一些无用单元回收开销是因为软件的瑕疵之外，存储器堆栈的使用没有优化和不正确的无用单元的回收策略是导致无用单元回收开销的主要原因。为优化存储器使用而调整堆栈和调整无用单元回收策略的程序即“无用单元回收的调整”。

无用单元回收的调整在一定的企业应用程序中通过优化存储器的使用和减少整个无用单元回收活动所花费的时间产生大量（如 100%，或相当高）的操作。这使得在 Java 或类似的程序语言中可以获得预计和优化的应用性能。

目前的无用单元回收工具作为成形的或监控解决方案被 Java 所整合。这些工具一般使用 Java 公用界面测量堆栈的容量，或依靠 JVM 所具有的无用单元回收跟踪功能。当这些工具提供给使用者优化无用单

元回收的指导时，它们同时也具有很多缺陷。更重要的是，这些工具不能提供任何真正的无用单元回收优化参数，提供解除下述昂贵操作的方法。

图 1 是说明根据现有技术无用单元回收调整过程 100 的流程图。在步骤 105 中，被用来调整无用单元回收的存储器分配跟踪特定为解释器。在步骤 110 中，应用程序执行写入应用程序中的程序语言。在步骤 115 中，跟踪数据被分析。在步骤 120 中，它判断所期望的结果是否已经达到。如果是，过程 100 结束。否则，如果所期望的结果没有达到，如步骤 125 所示，堆栈参数根据分析和过程 100 被调整，返回步骤 105。

一般来说，无用单元回收是作为用户键入或默认的参数。为了判断哪些参数要被插入，用户必须运行应用程序，并分析 Java 堆栈性能结果。例如，用户必须完成应用程序的运行而回收性能数据，并分析结果。当全部无用单元回收时间很短暂时，无用单元回收将成为大的应用程序而占用所有执行时间中相当长的一段，如企业应用程序。例如，在大的应用程序中，无用单元回收占用了所有的执行时间。当完成应用程序的运行后，分析堆栈性能数据，新的参数将由用户根据分析而确定。这样的过程将被重复，直至获得所期望的参数。

此外，为了优化无用单元回收，用户必须在应用程序或平台发生变化时，随时决定参数。运行应用程序、回收堆栈性能数据、分析结果所花费的时间可能相当大，这就导致了这些应用程序的运行有着不扉的开销。

当应用程序分析被诊断为不合规则时，额外的问题随之而生。安装，特别是在第三方的计算机系统上安装一个应用程序，一般的讲是麻烦的。而且，出于策略考虑，应用程序的拥有者或许不希望在第三方的计算机系统上安装其应用程序。于是，第三方必须告诉用户，哪些参数要键入，然后等待应用程序的运行，回收堆栈性能数据。这样就花费了应用程序用户的计算资源，或许超过了直接运用应用程序开销，增加了额外的开销（如，网络停机时间）和其它应用程序的性能下降。

发明内容

公开了一种优化计算机应用程序存储器使用的方法。接收包含计

时信息的存储器使用数据。产生存储器使用数据的图形表示。接收至少一个堆栈参数。并执行基于存储器使用数据和堆栈参数的存储器使用数据仿真。

附图说明

合并成为说明一部分的附图，将和说明一起用于说明本发明的实施方案，解释本发明的原理。

图 1 是说明现有无用数据回收整理技术的流程图。

图 2A 和图 2B 是说明根据本发明实施方案，无用数据回收整理选择参数的过程。

图 3 是根据本发明实施方案，由无用数据回收仿真图形用户界面产生的无用数据回收持续期视图的屏幕截图范例。

图 4 是根据本发明实施方案，由无用数据回收仿真图形用户界面产生的无用数据回收总结视图的屏幕截图范例。

图 5 是根据本发明实施方案，由无用数据回收仿真图形用户界面产生的无用数据回收堆栈参数输入视图的屏幕截图范例。

图 6 是根据本发明实施方案，说明仿真运行无用数据回收的流程图。

图 7 是根据本发明实施方案，说明数据在 Java 虚拟机和无用数据回收仿真间流动的数据流程图。

具体实施方式

本发明的首选实施方案的参考将与附图说明的示例予以详细的制定。本发明将和首选实施方案一道进行说明。可以理解，它不期望仅仅局限于本发明的这些实施方案中。相反，本发明意图含盖包括在附属权利声名所定义的本发明的思路和范围的所有变化、调整和等效。而且，在本发明下面的详细描述中，多数细节的提出是为了提供对本发明的一个详尽的理解。不管怎样，很明显，本发明技术领域的技术人员将无须这些细节而进行执行。在其它的实例中，为了避免不必要的模糊本发明的特征，众所周知的过程、构成、结构和装置将不再详细予以描述。

图 2A 和图 2B 是根据本发明实施方案的仿真程序用于存储器使用过程 200 的流程图。在实施方案中，过程 200 在计算机可读和计算机可执行的指令控制之下由处理器执行。计算机可读和计算机可执行指

令留驻在诸如数据存储特征、计算机非永久性存储器和/或计算机永久性存储器中。不管怎样，计算机可读和计算机可执行指令将留驻在计算机任何形式的可读介质中。尽管，特定的步骤在 200 过程中予以说明，这些步骤只是范例。就是说，本发明的实施方案非常适于执行图 2A 和图 2B 所描述的各种其它的步骤或变化。

在图 2A 的步骤 205 中，存储器分配性能跟踪被用作设定指定给解释器的无用数据回收。在一种实施方案中，解释器是 Java 虚拟机（JVM）。存储器分配性能跟踪是用作捕捉供执行离线仿真和执行应用最小指导的必须且充分信息的机构。在一种实施方案中，存储器分配性能跟踪在 HP-UX 上为-Xverbosegc 操作。在一种实施方案中，无用回收性能跟踪数据包括关于无用数据回收的信息，在这里同样是存储器使用数据。为了澄清说明，存储器使用数据这里还包括无用数据回收数据、存储器性能跟踪和详细的无用数据回收跟踪数据。通常，存储器使用数据在表格形式文件中包括关于事件的详细信息和在无用数据回收中发生的操作。在实施方案中，存储器使用数据包括，但不局限于：

为什么执行无用单元回收；

无用单元回收要占用多少时间；

无用单元回收发生的时间；

这个特定的无用单元回收的回收活动要占用多少存储器（如，这个特定的无用单元回收的效果）；

堆栈容量的详细信息；

在无用单元回收前，各个片断所占的部分；

在无用单元回收后，各个片断所占的部分；与

被认为是“老”的对象大概的年龄。

在一种实施方案中，其应用程序是基于 Java 的，存储器使用数据由在 JVM 无用单元回收器上选定的-Xverbosegc 操作产生。

在步骤 210 中，用于程序编写的程序语言执行一个应用程序。在一种实施方案中，应用是企业应用。在一种实施方案中，应用是基于用 Java 程序语言编写的 Java 应用。为了本申请的目的，解释器指基于 Java 编程语言的 JVM，然而，值得重视的是，本发明的目的在于使

用回收未引用的动态分配对象（如无用单元回收）来释放未使用存储器的任何编程语言。

在步骤 215 中，无用单元回收数据（如存储器分配性能跟踪数据、存储器使用数据、未引用的对象回收数据）被无用回收单元仿真所接收。在一种实施方案中，无用单元回收仿真留驻在 JVM 中，存储器使用数据可以直接访问。在另一种实施方案中，存储器使用数据通过 email 附件的形式发送到无用单元回收仿真。另一种实施方案中，存储器使用数据通过软盘复制文件进行发送。值得重视的是，存储器使用数据通常是小的文件，非常便于在计算机系统到计算机系统间传送。

在步骤 220 中，无用单元回收仿真被访问。无用单元回收仿真包括一个图形用户界面，提供了用于分析和调整堆栈参数、观察无用单元回收仿真结果的用户友好界面。在一种实施方案中，无用单元回收仿真留驻在如 JVM 同样的计算机系统中。在另一个实施方案中，无用单元回收仿真通过网络连接进行访问。

在步骤 225 中，存储器使用数据的图形显示被生成。如上所述，存储器使用数据为表格形式，包括绘制所产生的多个数据。在一种实施方案中，绘制产生的多个数据用于说明无用单元回收的不同方面。

特别的，在存储器使用数据捕获的计时信息用于绘制对应于应用程序所有执行时间的各种信息。图 3 是根据本发明的一种实施方案，在数据绘制 305 期间由无用单元回收仿真的图形用户界面所产生的无用单元回收范例的屏幕截图 300。

数据绘制 305 包括：所有应用程序以秒为单位运行时间的一个 X 轴 310 ，以秒为单位的特定的无用单元回收程序运行时间的 Y 轴 315。而且，数据绘制 305 说明的是运行时间的四种无用单元回收程序：提取、系统无用单元回收，老的全部和最老的全部。特别的，数据绘制 305 说明当无用单元回收发生时，什么类型的回收发生和它要运行多长的时间。所有计时的信息，如平均无用单元回收的提取持续时间、在计时信息 320 中所示的平均满无用单元回收持续时间。

参照图 2A 步骤 225，大范围的数据绘制或图形表示可以由存储器使用数据产生。在一种实施方案中，额外的数据绘制包括，但并不局限于：与时间对应的堆栈使用；与时间对应的自由累积字节；和形成率。本发明的实施方案直接允许用户定义数据的图形表示。用户期望

的特定无用单元回收信息可以基于要求的输入产生数据绘制。

在一种实施方案中，存储器使用数据使用文本格式来表示说明在应用程序运行期间，无用单元回收活动的总结。图 4 是根据本发明的一种实施方案，由无用单元回收仿真图形用户界面产生的总结视图 405 的示例屏幕截图 400。总结视图 405 说明的是关于 JVM 堆栈容量、无用单元回收活动和所有统计的详细信息。计时信息 410 包括花费在无用单元回收上时间的详细信息。饼形图 415 是所有运行在无用回收上时间方面的另一个图形表示。

参照图 2A 的步骤 225，一些无用单元回收信息的图形表示呈献给用户。使用这些图形表示，用户可以分析一个应用程序的无用单元回收活动。基于用图形表示所呈献的信息，用户可以为了优化应用程序的操作性能而设想改变各种输入参数。

在图 2B 的步骤 230 中，至少一个堆栈参数（如一个参数，一个存储器标准或堆栈输入）被无用单元回收仿真接收。在一种实施方案中，用户进入堆栈参数由无用单元回收仿真接收。更佳的，堆栈参数可以由计算机产生，本发明并不意图限制由用户产生堆栈参数的输入。

图 5 是根据本发明的一种实施方案，由无用单元回收仿真图形用户界面产生的堆栈参数输入视图 505 屏幕截图 500 的一个示例。堆栈参数输入视图允许用户通过改变各种堆栈参数而调整应用程序的无用单元回收。在一种实施方案中，有 4 个参数：全部堆栈大小 510、新产生的大小 515，存活率 520 和系统无用单元回收呼叫指示器 525。

在一种实施方案中，堆栈输入参数视图 505 表示当应用程序运行时存在的堆栈参数。然后用户参照原值来改变堆栈参数的值。例如，JVM 所有堆栈的大小 510 为 64M，堆栈参数输入视图相应的所有堆栈大小 510 说明值 64M。用户知道原值，然后就可以改变所有堆栈大小 510 的值。

在一种实施方案中，所有堆栈大小 510 的堆栈参数值、新产生大小 515 和存活率 520 可以直接由文本框（如文本框 512a-c）输入文本来改变。在一种实施方案中，所有堆栈大小 510 的堆栈参数值、新产生大小 515 和存活率 520 可以通过移动滚动条（如滚动条 514a-c）来改变。在一种实施方案中，文本框 512 中的值随着滚动条 514 的移动而更新。在另一种实施方案中，滚动条 514 随着在文本框 512 中输入新的值而更新。

在一种实施方案中，指示器 525 是一个包括在应用程序运行中、指示是否开始要求存在存储器管理的检查框。如果应用程序开始上述要求，根据跟踪数据，检查框开始检查。用户可以依据应用程序的特性选择检查框或不选择。

在一种实施方案中，堆栈参数输入视图 505 包括 JVM 选择器 530。在一种实施方案中，JVM 选择器是一个包括多个 JVM 版本的下拉菜单。JVM 选择器具有产生其它 JVM 版本/操作选项的功能，在开始说明哪些 JVM 版本/类型在应用程序运行中使用。用户可以通过选择不同的 JVM 版本改变必须重运行它们应用程序的 JVM 版本/类型选项。一旦用户键入所期望的参数，仿真通过激活仿真激励器 540 执行。

在一种实施方案中，堆栈参数输入视图 505 包括 JVM 选项 535。JVM 选项 535 是包括所选择所有堆栈大小 510 的堆栈参数值、新产生大小 515 和存活率 520 和系统无用单元回收呼叫指示器 525 的文本字符串。一旦所期望的堆栈参数被获得，JVM 操作 535 可以直接在 JVM 中进行复制。

在图 2B 的步骤 235 中，工作负荷属性被调整。更佳的是步骤 235 是可选的，工作负荷属性将保持一致。在步骤 240 中，无用单元回收仿真配置被调整。更佳的是步骤 240 是可选的，无用单元回收仿真配置将保持一致。

在图 2B 的步骤 245 中，无用单元回收仿真（如存储器使用仿真）被执行。在一种实施方案中，选择一个仿真激励器（如图 3 中的仿真激励器 340）激活无用单元回收仿真。无用单元回收仿真建立在存储器使用数据和输入参数的基础之上。

图 6 是说明根据本发明实施方案，仿真无用单元回收过程 600 的流程图。在步骤 605 中，详细的无用单元回收跟踪数据被接收。在一种实施方案中，无用单元回收跟踪数据通过在 HP-UX 的-Xverbosegc 操作来获得。在一种实施方案中，无用单元回收跟踪数据包括关于无用单元回收信息和与此相关的无用单元回收数据。

在步骤 610 中，它基于无用单元回收跟踪数据的格式来确定什么样的仿真模型。在步骤 615 中，基于输入的无用单元回收跟踪数据，确定源堆栈配置和策略。在步骤 620 中，至少对堆栈参数的一个调整被接收。更佳的是，调整将用于调整堆栈配置。工作负荷特性或仿真

参数。

在步骤 625 中，仿真特性由表示在跟踪数据中的每个测量间隔来确定。在一种实施方案中，仿真特性参数包括：

确定所有分配的存储器；

确定所有回收的存储器；

估算存储器分配率；和

根据存储器变为非引用的概论。

在步骤 630 中，新的应用运行根据由步骤 625 确定的仿真特性进行仿真。在一种实施方案中，无用单元回收仿真数据估算调整的堆栈配置包括，但并不局限于：

无用单元回收的间隔；

每单个回收的持续时间；

每次无用单元回收的起因；

两次回收间的所有存储器分配；

两次回收间的所有存储器回收；

每单次无用单元回收的持续时间；与

通过寿命来估算存储器使用的分布。

在如 2B 步骤 250 中，无用单元回收仿真数据的分析和图形表示被绘制。在一种实施方案中，无用单元回收仿真数据表示在表格形式中，因此适于生成多数据绘制。在一种实施方案中，多数据绘制被生成用作说明无用单元回收仿真的不同方面。特别的，无用单元回收仿真数据包括计时信息，允许对应应用程序执行的所有时间轴上绘制各种信息。

更佳的是，产生用作无用单元回收仿真数据的图形表示与在图 2A 的步骤 225 产生的无用单元回收数据相似。例如，图 3 的屏幕截图 300 示例同样是根据无用单元仿真数据的无用单元回收期间数据图 305 的说明。同样，图 4 的屏幕截图 400 示例同样是根据无用单元仿真数据的总结视图 405 的说明。

对无用单元数据回收，广泛的数据绘制或图形表示可以通过无用单元数据仿真数据来产生。在一种实施方案中，额外的数据包括，但不局限于：与时间对应的堆栈使用；与时间对应的自由累积字节；和形成率。本发明的一种实施方案直接允许用户定义数据的图形表示。

用户期望的特定无用单元回收信息可以基于要求输入产生数据绘制。

在一种实施方案中，对比无用单元回收仿真数据和无用单元回收数据产生一个图形表示。在另一种实施方案中，对比无用单元回收仿真数据和另一套无用单元回收仿真数据产生一个图形表示。更佳的是，任何数量的无用单元仿真的数据集间可以进行比较。对比的图形表示允许用户分析同一应用程序的多种不同的堆栈参数操作，而且允许用户选择想要的应用程序的堆栈参数集。

用户被呈献给一些无用单元数据仿真信息的图形表示。使用这些图形表示，用户可以分析一个用于程序的无用单元数据活动。根据呈献在图形表示中的信息，用户可以设想为了优化应用程序的性能而改变各种输入参数并返回仿真。

在步骤 255 中，它确定是否运行另一个仿真。在一个实施方案中，这个决定是由用户做出的。假设用户决定运行另一个仿真，程序 200 返回到步骤 230 中，且至少一个新的堆栈参数被输入。否则，假设用户不想运行另一个仿真，程序 200 继续执行到步骤 250。

在图 2A 的步骤 260 中，优化的堆栈参数可以插入到 JVM 中。在一种实施方案中，堆栈参数由无用单元回收仿真作为一个字符串提供（如图 5 的 JVM 选项 535）。在一种实施方案中，字符串由无用单元回收仿真直接插入到 JVM 中。在另一种实施方案中，优化由无用单元回收仿真直接复制并粘贴字符串到 JVM。

在步骤 265 中，它确定是否继续仿真结果的分析。在一种实施方案中，这个决定是由优化作出的。假定用户决定继续分析，程序 200 返回到步骤 210。否则，假设用户不想继续分析，程序 200 继续执行到步骤 270。

在步骤 270 中，具有优化的堆栈参数的应用程序在 JVM 中执行。在一种实施方案。更佳的是，步骤 270 是可选的，优化堆栈参数是可靠的。然而，因为优化堆栈参数是仿真的结果，希望通过运行有这优化堆栈参数的应用程序来确定其有效性。

图 7 是根据本发明的一种实施方案，说明数据在 JVM705 和存储器使用仿真 715（如无用单元回收仿真）间流动的流程图 700。更佳的是，JVM705 和存储器使用仿真 715 可以留驻在相同的计算机系统或不同的计算机系统中。在 JVM705 和存储器使用仿真 715（如存储器使用数据

720 和优化参数 745）间的数据传送通常很小，无论 JVM705 和存储器使用仿真 715 留驻在哪里，都非常便于传送。

基于 JAVA 的应用程序 710 在 JVM705 中执行，其中存储器使用数据 720 被产生。存储器使用数据 720 传送到存储器使用仿真 715。存储器使用数据 720 包含存储器使用信息，包括：与每个无用单元回收相关的计时信息。存储器使用仿真 715 产生存储器使用数据的图形表示 740。

响应图形表示 740，至少一个参数被输入到存储器使用仿真 715。可供选择的，工作负荷特性 730 和仿真配置 735 被输入到存储器使用仿真 715。根据存储器使用数据 720 和输入的堆栈参数 725，仿真被执行。特别的，根据存储器使用数据 720，一些假定被做出。例如，在无用单元回收间的线性概率被假设。通过假设无用单元回收的性能，仿真可以根据堆栈参数的变化而操作。更佳的是，本发明的其它实施方案可以做出不同的假定。

在一种实施方案中，一旦完成仿真，图形表示 740 根据仿真的结果而更新显示。更佳的是，仿真可以为堆栈参数 725 重复任何次。一旦决定堆栈参数是优化的，优化堆栈参数 745 可以插入到 JVM。

本发明的优选实施方案，一种优化计算机应用程序的存储器使用方法被描述。虽然本发明是以特定的实施方案来描述的，本发明并不局限在这些实施方案中，而是参照以下权利要求而解释的。

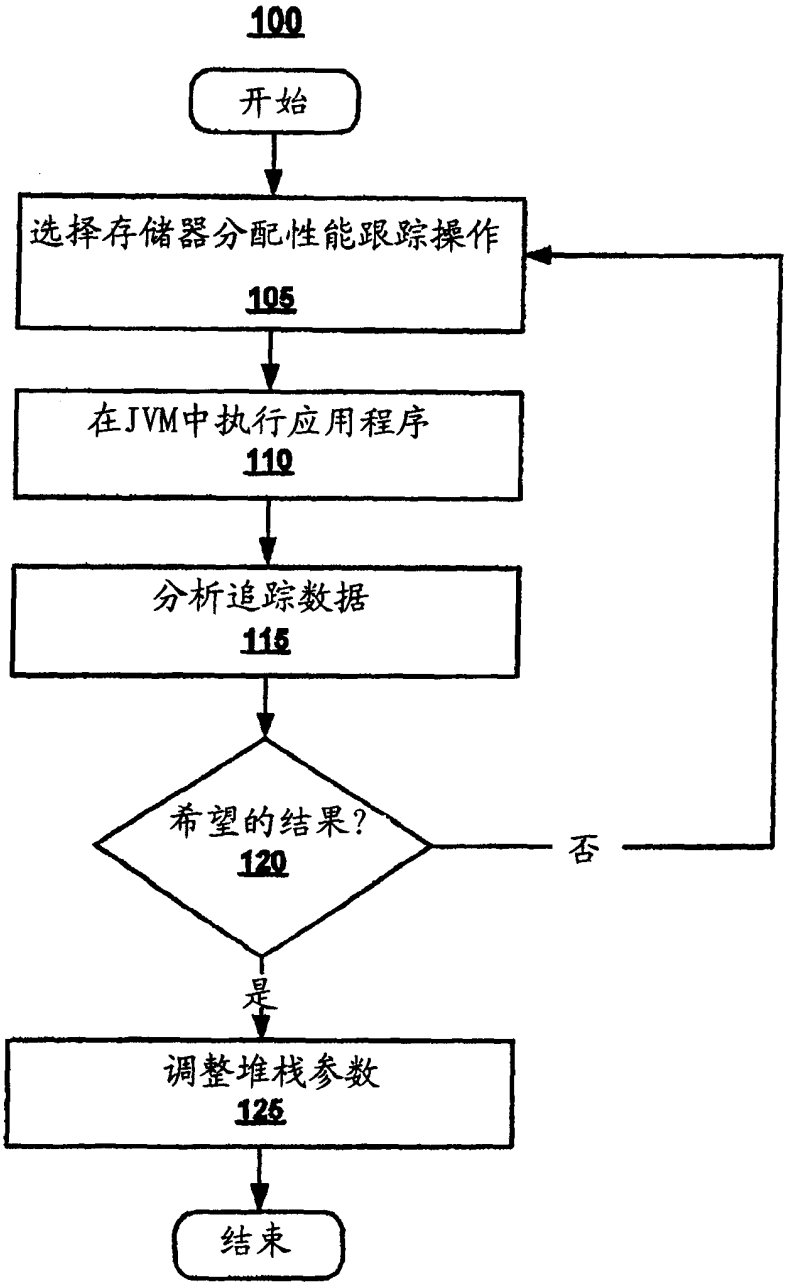

100
开始
选择存储器分配性能跟踪操作
105
在JVM中执行应用程序
110
分析追踪数据
115
希望的结果?
120
否
是
调整堆栈参数
125
结束

图 1

现有技术

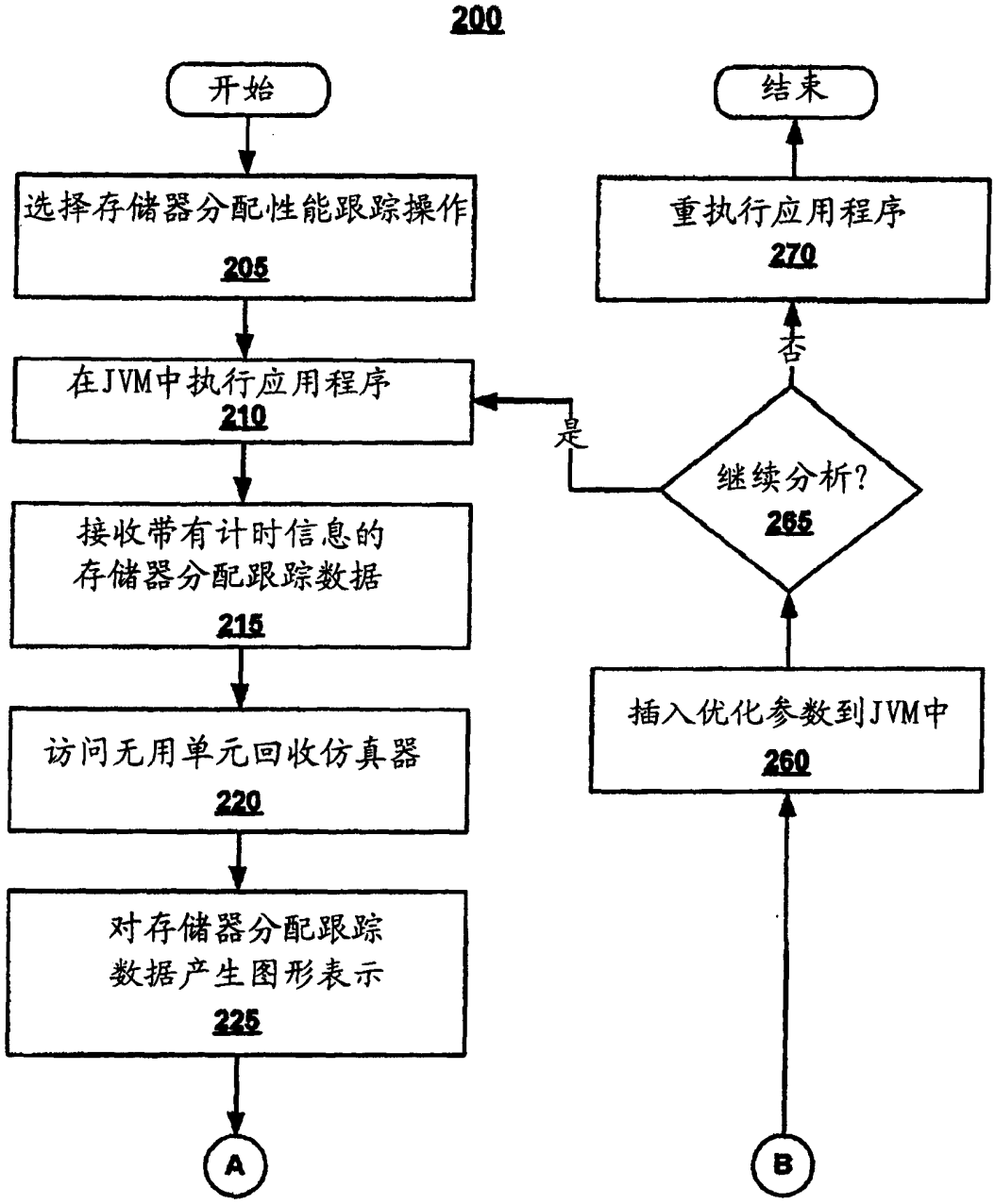

200
开始
选择存储器分配性能跟踪操作
205
在JVM中执行应用程序
210
接收带有计时信息的
存储器分配跟踪数据
215
访问无用单元回收仿真器
220
对存储器分配跟踪
数据产生图形表示
225
A
结束
重执行应用程序
270
否
是
继续分析?
265
插入优化参数到JVM中
260
B

图 2A

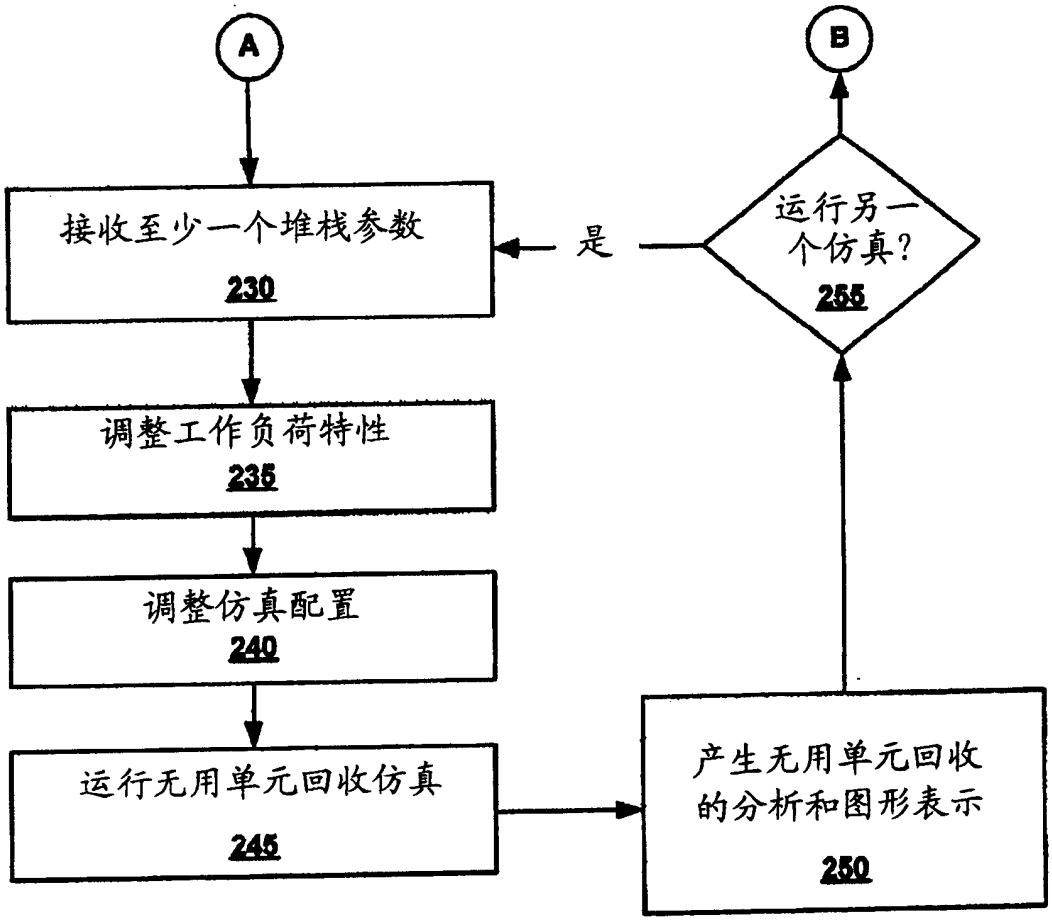

A
B
接收至少一个堆栈参数
230
是
运行另一个仿真?
255
调整工作负荷特性
235
调整仿真配置
240
运行无用单元回收仿真
245
产生无用单元回收的分析和图形表示
250

图 2B

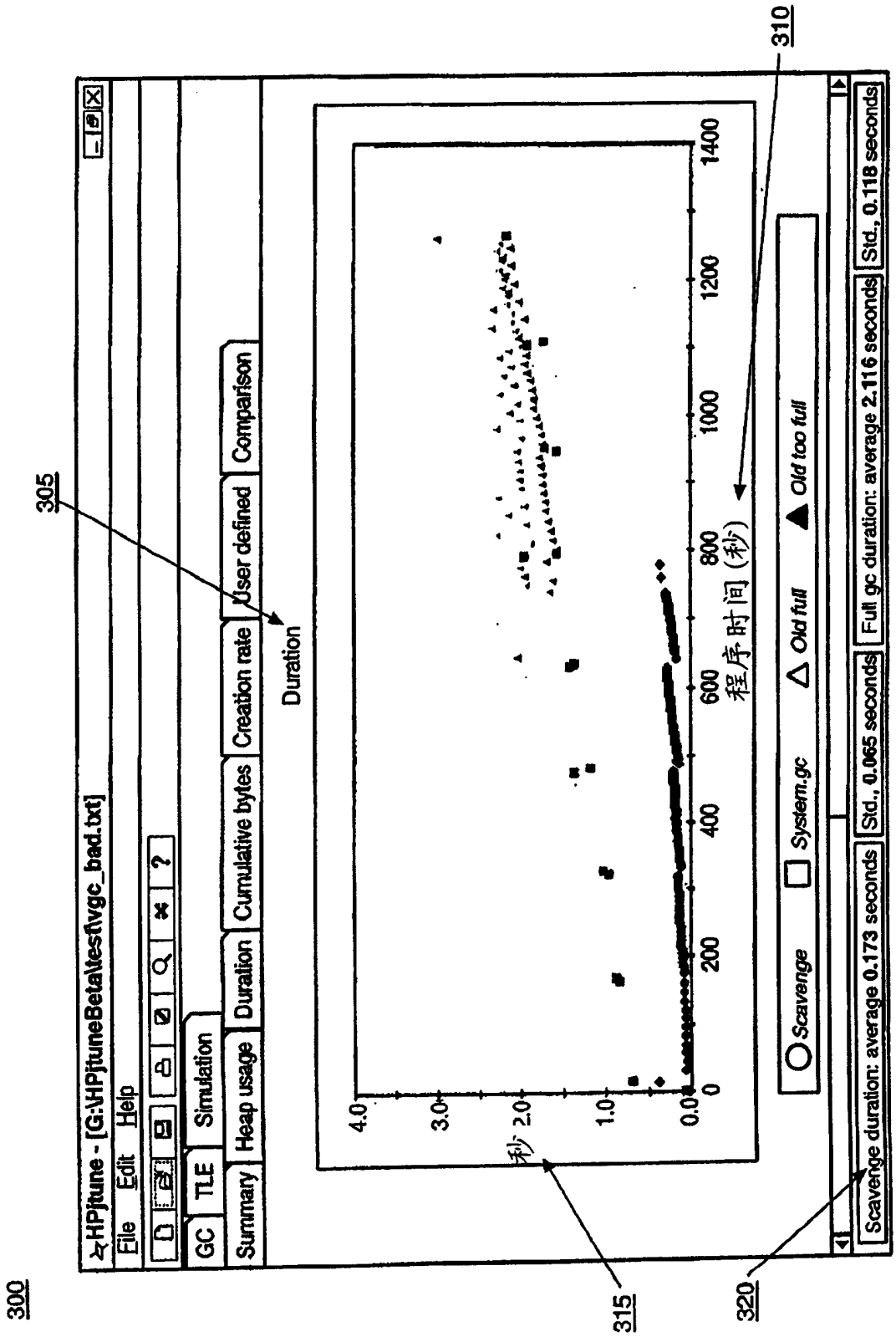

300
305
310
315
320
HPjtune - [G:\HPjtuneBeta\test\vgc_bad.txt]
File Edit Help
GC
TLE
Simulation
Summary
Heap usage
Duration
Cumulative bytes
Creation rate
User defined
Comparison
Duration
4.0
3.0
2.0
1.0
0.0
秒
0
200
400
600
800
1000
1200
1400
程序时间(秒)
Scavenge
System.gc
Old full
Old too full
Scavenge duration: average 0.173 seconds
Std., 0.065 seconds
Full gc duration: average 2.116 seconds
Std., 0.118 seconds

图 3

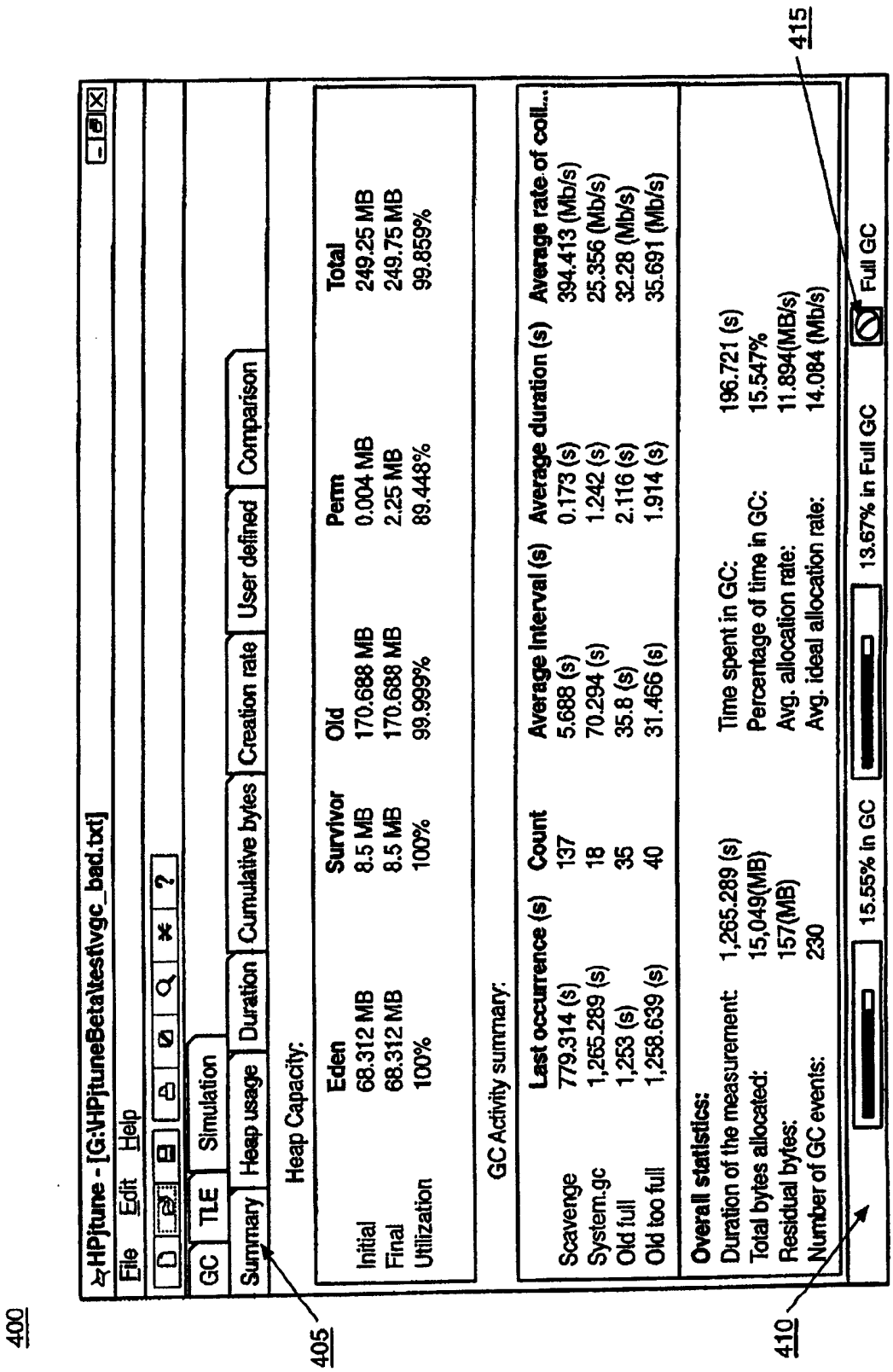

400
HPjtune - [G:\HPjtuneBeta\test\vgc_bad.txt]
File Edit Help
GC TLE Simulation
Summary Heap usage Duration Cumulative bytes Creation rate User defined Comparison
405
Heap Capacity:
Eden Survivor Old Perm Total
Initial 68.312 MB 8.5 MB 170.688 MB 0.004 MB 249.25 MB
Final 68.312 MB 8.5 MB 170.688 MB 2.25 MB 249.75 MB
Utilization 100% 100% 99.999% 89.448% 99.859%
GC Activity summary:
Last occurrence (s) Count Average interval (s) Average duration (s) Average rate of coll...
Scavenge 779.314 (s) 137 5.688 (s) 0.173 (s) 394.413 (Mb/s)
System.gc 1,265.289 (s) 18 70.294 (s) 1.242 (s) 25.356 (Mb/s)
Old full 1,253 (s) 35 35.8 (s) 2.116 (s) 32.28 (Mb/s)
Old too full 1,258.639 (s) 40 31.466 (s) 1.914 (s) 35.691 (Mb/s)
Overall statistics:
Duration of the measurement: 1,265.289 (s)
Total bytes allocated: 15,049(MB)
Residual bytes: 157(MB)
Number of GC events: 230
Time spent in GC: 196.721 (s)
Percentage of time in GC: 15.547%
Avg. allocation rate: 11.894(MB/s)
Avg. ideal allocation rate: 14.084 (Mb/s)
410
15.55% in GC
13.67% in Full GC
Full GC
415

图 4

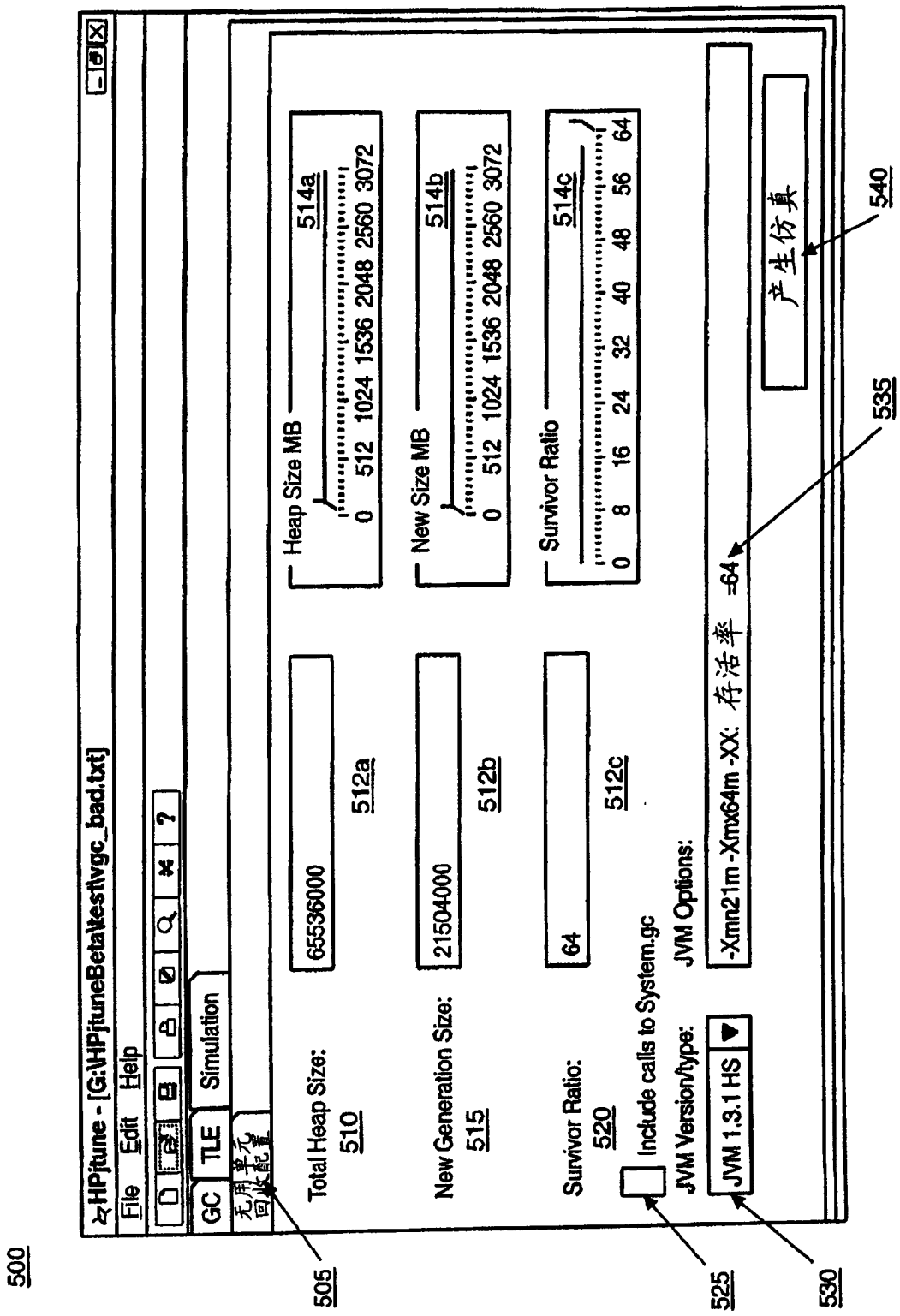

500
HPjtune - [G:\HPjtuneBeta\test\gc_bad.txt]
File Edit Help
GC
TLE
Simulation
无用单元回收配置
505
Total Heap Size:
510
65536000
512a
New Generation Size:
515
21504000
512b
Survivor Ratio:
520
64
512c
Include calls to System.gc
525
JVM Version/type:
JVM 1.3.1 HS
530
JVM Options:
-Xmn21m -Xmx64m -XX: 存活率 =64
535
Heap Size MB
514a
0 512 1024 1536 2048 2560 3072
New Size MB
514b
0 512 1024 1536 2048 2560 3072
Survivor Ratio
514c
0 8 16 24 32 40 48 56 64
产生仿真
540

图 5

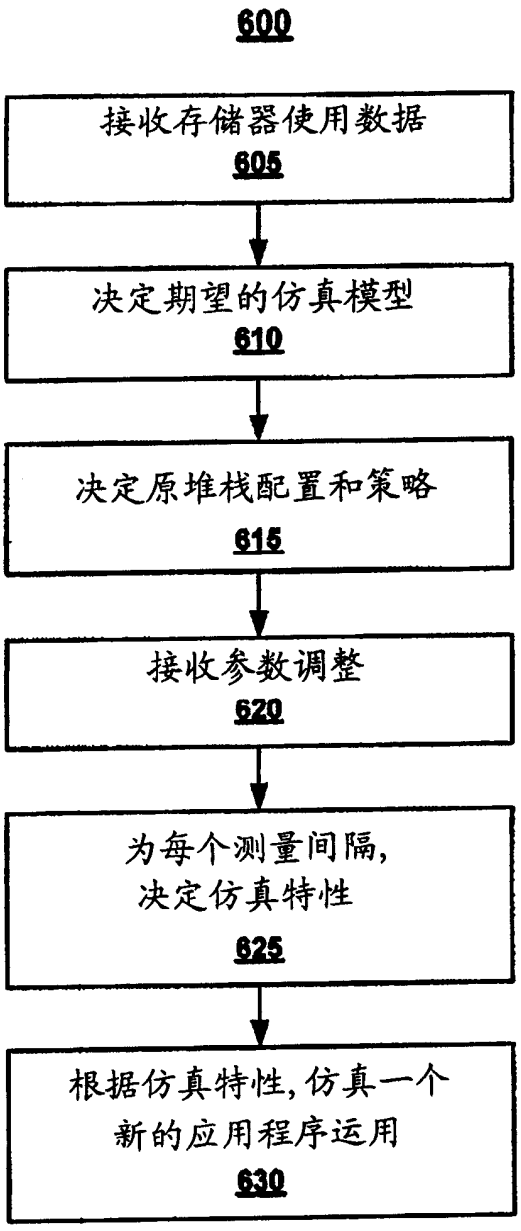

600
接收存储器使用数据
605
决定期望的仿真模型
610
决定原堆栈配置和策略
615
接收参数调整
620
为每个测量间隔，
决定仿真特性
625
根据仿真特性，仿真一个
新的应用程序运用
630

图 6

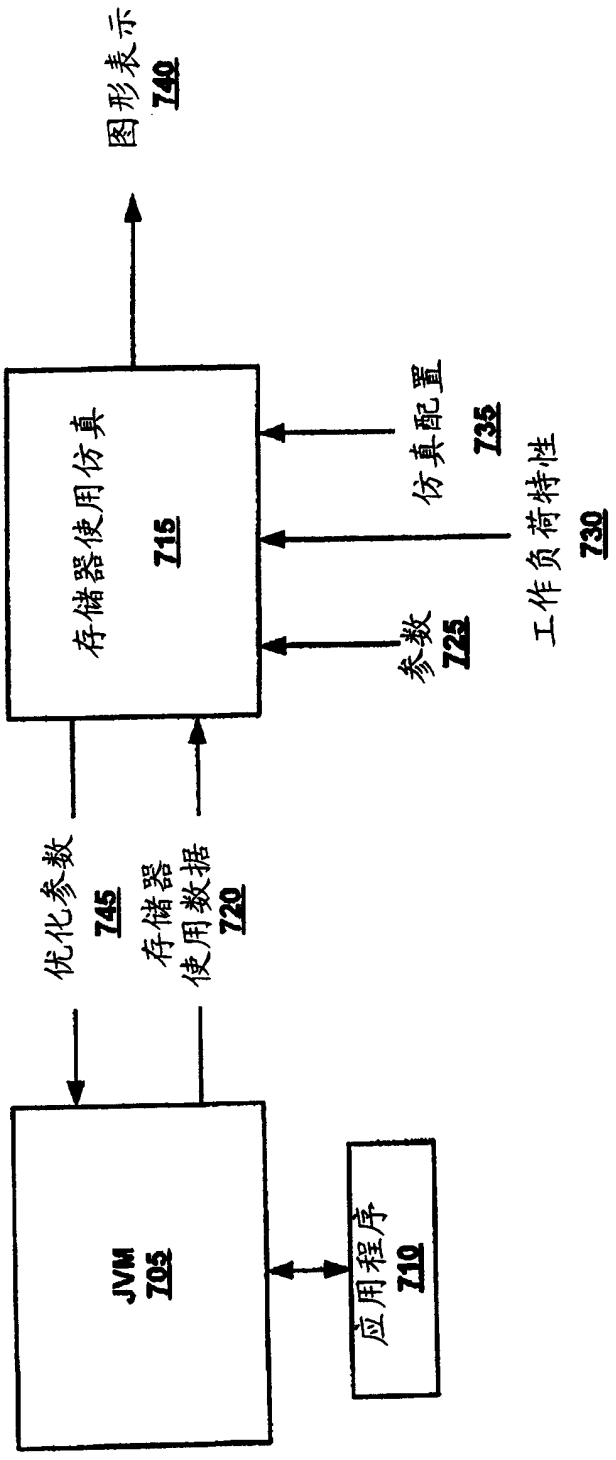

图形表示
740
存储器使用仿真
715
仿真配置
735
工作负荷特性
730
参数
725
优化参数
745
存储器
使用数据
720
JVM
705
应用程序
710

图 7