(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 106933689 B
(45)授权公告日 2020.05.19

(21)申请号 201511018824.2

(22)申请日 2015.12.29

(65)同一申请的已公布的文献号
申请公布号 CN 106933689 A

(43)申请公布日 2017.07.07

(73)专利权人 伊姆西IP控股有限责任公司
地址 美国马萨诸塞州

(72)发明人 肖会兵 高健 韩耕 董继炳 高宏坡

(74)专利代理机构 北京市金杜律师事务所 11256
代理人 王茂华 张曦

(51)Int.Cl.
G06F 11/07(2006.01)

(56)对比文件
US 2003005414 A1,2003.01.02,
CN 101719090 A,2010.06.02,
CN 102597962 A,2012.07.18,
CN 1983206 A,2007.06.20,
CN 103164322 A,2013.06.19,

审查员 姜晓庆

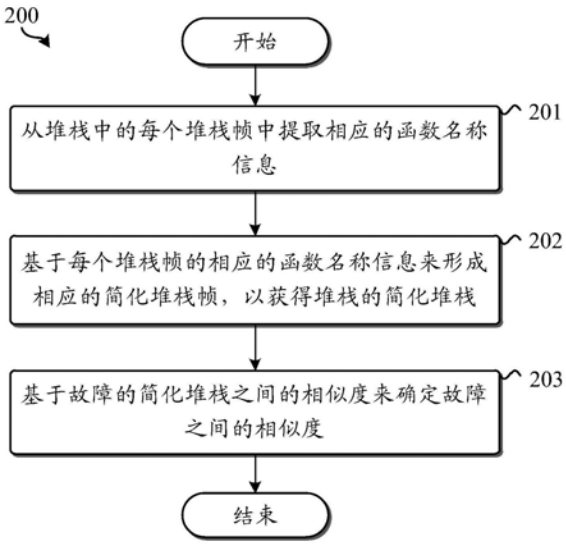
权利要求书3页 说明书11页 附图4页

(54)发明名称

一种用于计算设备的方法和装置

(57)摘要

本公开内容的实施例提供了一种用于计算设备的方法和装置。该计算设备可以在每次发生故障时生成用于故障转储的堆栈，每个堆栈从底部到顶部可以包括多个堆栈帧，每个堆栈帧可以包括与故障有关的函数信息。该方法可以包括：从堆栈中的每个堆栈帧中提取相应的函数名称信息；基于每个堆栈帧的相应的函数名称信息来形成相应的简化堆栈帧，以获得堆栈的简化堆栈；以及基于故障的简化堆栈之间的相似度来确定故障之间的相似度。



CN 106933689 B

1.一种用于计算设备的方法,所述计算设备在每次发生故障时生成用于故障转储的堆栈,每个堆栈从底部到顶部包括多个堆栈帧,每个堆栈帧包括与故障有关的函数信息,所述方法包括:

从所述堆栈中的每个堆栈帧中提取相应的函数名称信息;

基于每个堆栈帧的相应的函数名称信息来形成相应的简化堆栈帧,以获得所述堆栈的简化堆栈;以及

通过以下而基于故障的简化堆栈之间的相似度来确定故障之间的相似度:

从简化堆栈底部的简化堆栈帧开始向顶部方向逐个地移除简化堆栈帧直到仅剩余顶部的简化堆栈帧,每移除一个简化堆栈帧生成一个子堆栈;

基于简化堆栈之间的相似度和简化堆栈的子堆栈之间的相似度来确定故障之间的相似度;

针对多次故障中的每次故障来获得相应的简化堆栈和简化堆栈的子堆栈;以及

形成所述多次故障的简化堆栈和子堆栈与所述多次故障之间的对应关系表格。

2.根据权利要求1所述的方法,其中基于每个堆栈帧的相应的函数名称信息来形成相应的简化堆栈帧包括:

仅利用每个堆栈帧的相应的函数名称信息来形成相应的简化堆栈帧。

3.根据权利要求1所述的方法,其中基于每个堆栈帧的相应的函数名称信息来形成相应的简化堆栈帧包括:

利用每个堆栈帧的相应的函数名称信息和相应的模块名称信息来形成相应的简化堆栈帧。

4.根据权利要求3所述的方法,进一步包括:

从每个堆栈帧中提取相应的模块名称信息,或者通过查找每个堆栈帧中的函数地址信息来确定相应的模块名称信息。

5.根据权利要求3所述的方法,进一步包括:

在相应的简化堆栈帧中,在所述相应的函数名称信息与所述相应的模块名称信息之间设置分隔符。

6.根据权利要求1所述的方法,进一步包括:

基于故障之间所共有的简化堆栈或者具有最多简化堆栈帧的子堆栈来确定故障之间的相似度。

7.根据权利要求1所述的方法,其中基于故障的简化堆栈之间的相似度来确定故障之间的相似度包括:

针对新故障来获得相应的简化堆栈及其子堆栈;以及

在所述对应关系表格中查找所述新故障的简化堆栈及其子堆栈,以确定与所述新故障具有相同简化堆栈或子堆栈的故障。

8.根据权利要求7所述的方法,进一步包括:

将所述新故障的不在所述对应关系表格中的简化堆栈和子堆栈添加到所述对应关系表格中。

9.根据权利要求1所述的方法,进一步包括:

使用哈希算法来形成所述对应关系表格。

10.一种用于计算设备的装置,所述计算设备在每次发生故障时生成用于故障转储的堆栈,每个堆栈从底部到顶部包括多个堆栈帧,每个堆栈帧包括与故障有关的函数信息,所述装置包括:

提取单元,被配置为从所述堆栈中的每个堆栈帧中提取相应的函数名称信息;

简化单元,被配置为基于每个堆栈帧的相应的函数名称信息来形成相应的简化堆栈帧,以获得所述堆栈的简化堆栈;

相似度确定单元,被配置为基于故障的简化堆栈之间的相似度来确定故障之间的相似度;

子堆栈生成单元,被配置为从简化堆栈底部的简化堆栈帧开始向顶部方向逐个地移除简化堆栈帧直到仅剩余顶部的简化堆栈帧,每移除一个简化堆栈帧生成一个子堆栈;

其中所述相似度确定单元进一步被配置为:基于简化堆栈之间的相似度和简化堆栈的子堆栈之间的相似度来确定故障之间的相似度;

其中所述简化单元进一步被配置为:针对多次故障中的每次故障来获得相应的简化堆栈和简化堆栈的子堆栈;并且

其中所述装置进一步包括:对应关系表格形成单元,被配置为形成所述多次故障的简化堆栈和子堆栈与所述多次故障之间的对应关系表格。

11.根据权利要求10所述的装置,其中所述简化单元进一步被配置为:

仅利用每个堆栈帧的相应的函数名称信息来形成相应的简化堆栈帧。

12.根据权利要求10所述的装置,其中所述简化单元进一步被配置为:

利用每个堆栈帧的相应的函数名称信息和相应的模块名称信息来形成相应的简化堆栈帧。

13.根据权利要求12所述的装置,其中所述简化单元进一步被配置为:

从每个堆栈帧中提取相应的模块名称信息,或者通过查找每个堆栈帧中的函数地址信息来确定相应的模块名称信息。

14.根据权利要求12所述的装置,其中所述简化单元进一步被配置为:

在相应的简化堆栈帧中,在所述相应的函数名称信息与所述相应的模块名称信息之间设置分隔符。

15.根据权利要求10所述的装置,其中所述简化单元进一步被配置为:

基于故障之间所共有的简化堆栈或者具有最多简化堆栈帧的子堆栈来确定故障之间的相似度。

16.根据权利要求10所述的装置,其中所述简化单元进一步被配置为:

针对新故障来获得相应的简化堆栈及其子堆栈;并且

所述相似度确定单元进一步被配置为:在所述对应关系表格中查找所述新故障的简化堆栈及其子堆栈,以确定与所述新故障具有相同简化堆栈或子堆栈的故障。

17.根据权利要求16所述的装置,进一步包括:

添加单元,被配置为将所述新故障的不在所述对应关系表格中的简化堆栈和子堆栈添加到所述对应关系表格中。

18.根据权利要求10所述的装置,其中所述对应关系表格形成单元进一步被配置为:

使用哈希算法来形成所述对应关系表格。

19.一种计算机可读存储介质，具有存储在其上的计算机可读程序指令，所述计算机可读程序指令用于执行根据权利要求1-9中任一项所述的方法。

20.一种计算机系统，包括根据权利要求10-18中任一项所述的装置。

一种用于计算设备的方法和装置

技术领域

[0001] 本公开内容的实施例一般性地涉及与计算设备有关的技术领域，并且更特别地涉及一种用于计算设备的方法和装置。

背景技术

[0002] 在计算设备的操作过程中，当异常事件或意外情况发生时，用户或者系统可以进行故障转储(crash dump)以保存有用的上下文信息。在故障转储中，用于故障转储的堆栈是故障转储的最重要的信息或签名之一，并且指示了计算系统/进程发生故障的直接原因。针对故障转储堆栈进行堆栈帧的回溯可以给出导致了故障转储发生的唯一的调用代码路径的明确序列。

[0003] 对于处在繁重测试之下的系统而言，可能会生成许多具有类似堆栈的故障转储，并且可能经常会需要确定新的故障转储是否与正在被分析或者已经被分析过的其他故障转储有关，从而能够通过参考对其他故障转储的分析来避免重复的工作。

[0004] 但是，这样的确定通常并不是容易的事情，因为即使是在用于故障转储的堆栈具有相似的代码路径时，也几乎不可能找到完全匹配的故障转储。这是因为故障转储堆栈中总是存在许多差异或噪声。对于一些其他的外部社区而言，情况也是这样。例如，当使用流行的具有强大搜索引擎(例如，谷歌、百度等)的网页浏览器来搜索共同的完整Linux/Windows开源应用或内核故障转储堆栈时，通常不能找到任何有用的结果。

[0005] 在现有的解决方案中，因为每个个体的故障转储堆栈的完整文本通常包含许多噪声信息，所以为了确定两个故障转储之间的相似度，可能需要利用经典的分类算法(例如，贝叶斯分类算法)来构建专门定制的全文搜索引擎。

[0006] 然而，这种方法的缺点是它引入了许多噪声(由于来自堆栈文本的变化部分)，并且还丢失了堆栈中的调用次序信息。即使后者能够通过考虑堆栈文本中的词语的排序或次序来修改，但是如果故障转储的数目非常大，则计算成本也会相应地增大。因为需要将给定的堆栈文本与所有的已有堆栈文件进行比较。

[0007] 此外，这种方法还引入了非常大的复杂度并且使其难以在资源有限的计算机工作站上实施和快速运行。另外，按这种方法得出的相似度不能给出对与故障有关的问题的简单理解，因为它仅算入了个体的词语并且丢失了完整的上下文信息。

发明内容

[0008] 鉴于现有技术中存在的上述问题，本公开内容的实施例的目的之一在于提供一种用于计算设备的方法和装置，以解决现有技术中的上述以及其他的问题。

[0009] 根据本公开内容的第一方面，提供了一种用于计算设备的方法。该计算设备可以在每次发生故障时生成用于故障转储的堆栈，每个堆栈从底部到顶部可以包括多个堆栈帧，每个堆栈帧可以包括与故障有关的函数信息。该方法可以包括：从堆栈中的每个堆栈帧中提取相应的函数名称信息；基于每个堆栈帧的相应的函数名称信息来形成相应的简化堆

栈帧，以获得堆栈的简化堆栈；以及基于故障的简化堆栈之间的相似度来确定故障之间的相似度。

[0010] 根据本公开的一些实施例，基于每个堆栈帧的相应的函数名称信息来形成相应的简化堆栈帧可以包括：仅利用每个堆栈帧的相应的函数名称信息来形成相应的简化堆栈帧。

[0011] 根据本公开的一些实施例，基于每个堆栈帧的相应的函数名称信息来形成相应的简化堆栈帧可以包括：利用每个堆栈帧的相应的函数名称信息和相应的模块名称信息来形成相应的简化堆栈帧。

[0012] 根据本公开的一些实施例，该方法可以进一步包括：从每个堆栈帧中提取相应的模块名称信息，或者通过查找每个堆栈帧中的函数地址信息来确定相应的模块名称信息。

[0013] 根据本公开的一些实施例，该方法可以进一步包括：在相应的简化堆栈帧中，在相应的函数名称信息与相应的模块名称信息之间设置分隔符。

[0014] 根据本公开的一些实施例，基于故障的简化堆栈之间的相似度来确定故障之间的相似度包括：从简化堆栈底部的简化堆栈帧开始向顶部方向逐个地移除简化堆栈帧直到仅剩余顶部的简化堆栈帧，每移除一个简化堆栈帧生成一个子堆栈；以及基于故障的简化堆栈及其子堆栈之间的相似度来确定故障之间的相似度。

[0015] 根据本公开的一些实施例，该方法进一步包括：基于故障之间所共有的简化堆栈或者具有最多简化堆栈帧的子堆栈来确定故障之间的相似度。

[0016] 根据本公开的一些实施例，该方法可以进一步包括：针对多次故障中的每次故障来获得相应的简化堆栈及其子堆栈；以及形成多次故障的所有简化堆栈和所有子堆栈与多次故障之间的对应关系表格。

[0017] 根据本公开的一些实施例，基于故障的简化堆栈之间的相似度来确定故障之间的相似度可以包括：针对新故障来获得相应的简化堆栈及其子堆栈；以及在对应关系表格中查找新故障的简化堆栈及其子堆栈，以确定与新故障具有相同简化堆栈或子堆栈的故障。

[0018] 根据本公开的一些实施例，该方法可以进一步包括：将新故障的不在对应关系表格中的简化堆栈和子堆栈添加到对应关系表格中。

[0019] 根据本公开的一些实施例，该方法可以进一步包括：使用哈希算法来形成对应关系表格。

[0020] 根据本公开内容的第二方面，提供了一种用于计算设备的装置。该计算设备可以在每次发生故障时生成用于故障转储的堆栈，每个堆栈从底部到顶部可以包括多个堆栈帧，每个堆栈帧可以包括与故障有关的函数信息。该装置可以包括：提取单元，被配置为从堆栈中的每个堆栈帧中提取相应的函数名称信息；简化单元，被配置为基于每个堆栈帧的相应的函数名称信息来形成相应的简化堆栈帧，以获得堆栈的简化堆栈；以及相似度确定单元，被配置为基于故障的简化堆栈之间的相似度来确定故障之间的相似度。

[0021] 根据本公开内容的第三方面，提供了一种计算机可读存储介质，具有存储在其上的计算机可读程序指令，该计算机可读程序指令用于执行根据第一方面的方法。

[0022] 根据本公开内容的第四方面，提供了一种计算机系统，包括根据第二方面的装置。

[0023] 本公开内容的实施例提供了一种具有低复杂度的快速方式来确定给定的故障转储堆栈是否与另一故障转储堆栈具有较高的相似度，并且如果存在这样的故障转储堆栈，

则可以得出最佳的故障转储列表，而排除故障转储堆栈中的差异或者噪声的影响。

[0024] 本公开内容的实施例提出的方法是高效的并且其确定结果能够更为准确地给出对故障转储的之间的相似度。利用所提出的方法和装置，能够高效地找出相似的故障转储堆栈而无需针对要搜索的堆栈文本构建定制的全文搜索引擎，并且通过按照故障转储堆栈中的原有堆栈帧的时间顺序而逐个地删除堆栈帧来形成多个子堆栈，从而所形成的子堆栈不会丢失原有堆栈帧中的关键时序信息。此外，故障之间的相似度能够容易地被量化以筛选用于其他分析。

附图说明

[0025] 通过参考附图阅读下文的详细描述，本公开内容的实施例的上述以及其他目的、特征和优点将变得容易理解。在附图中，以示例性而非限制性的方式示出了本公开内容的若干实施例，其中：

[0026] 图1示意性地示出了一种典型的由原始堆栈帧组成的用于故障转储的堆栈。

[0027] 图2示意性地示出了根据本公开内容的实施例的方法的流程图。

[0028] 图3示意性地示出了根据本公开内容的实施例的装置的框图。

[0029] 图4示意性地示出了可以用来实现本公开内容的实施例的示例性计算机系统/服务器的框图。

具体实施方式

[0030] 下面将参考附图中所示出的若干示例性实施例来描述本公开内容的原理和精神。应当理解，描述这些具体的实施例仅是为了使本领域的技术人员能够更好地理解并实现本公开，而并非以任何方式限制本公开内容的范围。

[0031] 图1示意性地示出了一种典型的由原始堆栈帧组成的故障转储堆栈100。如上文所提到的，在发生故障转储时，用于故障转储的堆栈是故障转储的最重要的信息或签名之一，并且指示了计算系统/进程发生故障的直接原因。针对故障转储堆栈进行堆栈帧的回溯可以给出导致了故障转储发生的唯一的调用代码路径的明确序列。

[0032] 如图1中所示出的，示例性的故障转储堆栈100从底部到顶部可以包括编号#31至#0的原始堆栈帧，每个原始堆栈帧中可以包括与故障有关的函数信息。例如，编号为#0的堆栈帧包括与故障有关的函数raise的相关信息。为了清楚的原因，图1中仅示例性地示出了原始堆栈帧#0至#5以及#28至#31，并且省略了编号为#6至#27的原始堆栈帧。本领域的技术人员可以理解，编号为#6至#27的原始堆栈帧可以与所示出的原始堆栈帧具有相类似的结构和内容。

[0033] 原始的故障转储堆栈100可以给出与故障堆栈代码路径有关的完全细节信息，但是通过它难以发现许多故障转储的堆栈之间的相似度，因为堆栈回溯的每个堆栈帧中包含的大部分内容在每次对程序/进程进行编译、加载和分析时总是会发生变化。例如，故障转储堆栈100中包括带下划线的部分，这些部分与堆栈帧的地址相关并且在每次进行模块加载时可能会变化，代码行和目录信息在每个实时系统或者排错系统上下文中都可能会变化。

[0034] 在进行与计算设备的有关操作中，可能经常会需要确定新的故障转储是否与正在

被分析或者已经被分析过的其他故障转储之间具有较高的相似度，从而能够通过参考对其他故障转储的分析来避免重复的工作。这就需要一种高效而简单的方式以在可能生成许多故障转储(例如，百万级)的大产出系统中找出相似的故障转储堆栈，并且找出相似度最高的故障转储堆栈同时结果能够容易地被理解。

[0035] 从算法的视角来看，这是一种纯文本的相似度计算问题。通常，搜索引擎或者类似工具能够被用来首先根据文本中的词语来对堆栈文本进行索引，并且然后在搜索给定的堆栈文本的过程中计算给定的堆栈文本的词语矢量与所有其他故障转储堆栈的词语矢量之间的余弦相似度。如前文所论述的，现有技术中的方法存在各种缺陷。

[0036] 本公开内容的各种实施例提供了一种具有低复杂度的快速方式来确定给定的故障转储堆栈是否与另一故障转储堆栈具有较高的相似度，并且如果存在这样的故障转储堆栈，则可以得出最佳的故障转储列表，而排除故障转储堆栈中的差异或者噪声的影响。

[0037] 本公开内容的各个实施例提出的方法是高效的并且其确定结果能够体现出对故障转储的良好见解。利用所提出的方法，能够高效地找出相似的故障转储堆栈而无需针对要搜索的堆栈文本构建定制的全文搜索引擎。

[0038] 图2示意性地示出了根据本公开内容的实施例的方法200的流程图。在一些实施例中，方法200可以用于计算设备，该计算设备在每次发生故障时可以生成用于故障转储的堆栈，每个堆栈从底部到顶部可以包括多个堆栈帧，每个堆栈帧可以包括与故障有关的函数信息。在一些实施例中，方法200的执行主体可以是与计算设备有关的实体。在一些特定的实施例中，方法200的执行主体可以是稍后结合附图3所描述的装置300。

[0039] 如图2中所示出的，方法200在开始之后可以进入步骤201。在步骤201中，方法200的执行主体可以从堆栈中的每个堆栈帧中提取相应的函数名称信息。

[0040] 在特定的实施例中，可以选择引起故障转储的原始堆栈帧并且将堆栈帧进行如下的处理。首先，从每个原始堆栈帧中去除总是发生变化且不太重要的信息，而仅保留每个原始堆栈帧的函数名称信息。例如，这些可去除的信息可以包括：原始堆栈中的数字，诸如函数地址，它们在每次模块加载时将会改变，以及利用不同上下文的运行时间传递的参数，以及编译时间期间的源代码目录和行信息，等等。

[0041] 在一些具体的示例性实施例中，针对附图1中所示出的堆栈100，可以从编号为#0-#5的堆栈帧中分别提取出相应的函数名称信息：raise、abort、csx_rt_proc_do_abort、csx_rt_assert_int_take_user_space_panic_action、csx_rt_assert_request_panic_with_info、EmcpalBugCheck。进一步地，可以从编号为#28-#31的堆栈帧中分别提取出相应的函数名称信息：csx_rt_cpi_thread_command、csx_rt_sked_thread_wrapper、start_thread、clone。

[0042] 接着，方法200可以进入步骤202。在步骤202中，方法200的执行主体可以基于每个堆栈帧的相应的函数名称信息来形成相应的简化堆栈帧，以获得堆栈的简化堆栈。

[0043] 在一些实施例中，可以仅利用每个堆栈帧的相应的函数名称信息来形成相应的简化堆栈帧。本领域的技术人员可以理解，在故障转储堆栈的堆栈帧中，与故障有关的函数名称信息可以较好地体现两次故障之间的关联性。因此，在方法200的一种简单的实施方式中，可以仅使用每个堆栈帧中所包含的函数名称信息来形成每个堆栈帧的简化堆栈帧，保留故障转储堆栈中的各个堆栈帧的原有时序，使用各个堆栈帧的相应简化堆栈帧得出故障

堆栈的简化堆栈。

[0044] 在另一些实施例中，也可以利用每个堆栈帧的相应的函数名称信息和相应的模块名称信息来形成相应的简化堆栈帧。然后，通过保留故障转储堆栈中的各个堆栈帧的原有时序，使用各个堆栈帧的相应简化堆栈帧得出故障堆栈的简化堆栈。在这些实施例中，简化堆栈帧中除了包括函数名称信息之外，还可以包括相应的模块名称信息。这经常是有用的，因为相同的函数可能静态地联系到不同的模块，并且区分是哪个模块在堆栈中存在问题可能是有利的。

[0045] 可以使用不同的方式来获得与故障函数相关联的模块名称信息。在一些实施例中，如果堆栈帧中包括了相关的模块名称信息，则方法200可以包括：从每个堆栈帧中提取相应的模块名称信息。

[0046] 如果模块名称信息不能从堆栈帧中得到，这通常是对于校验/排错版本构建的情况。在一些实施例中，可以通过查找每个堆栈帧中的函数地址信息来确定相应的模块名称信息。具体而言，能够通过在用于该模块的故障转储地址中查找函数地址，然后从一些其他来源的中推导出该函数地址属于哪个模块。

[0047] 在一个具体的示例性实施例中，对于图1中所示出的堆栈帧#2，可以看出其函数地址为0x00007f6f74ac04d7并且其函数名称信息为csx_rt_proc_do_abort。基于这些信息，通过对所有的故障转储地址空间进行故障转储，能够确定该函数地址0x00007f6f74ac04d7落在范围0x00007f6f74a13cc0-0x00007f6f74b5bab8中。通过查询可知，该范围是csx_urt.so的.text区域。这一过程可以用代码表示如下。

[0048] (gdb)info files

[0049] Symbols from“/disks/USD_dumps15/ARs/0564000-0564999/564217/10-11-2013/EMC/csx/ubin64/csx_ic_std.x”.

[0050] Local core dump file:

[0051] ‘/tmp/abcdt/a/564217-spa_FNM00131203264_2013-10-11_17_14_08_8666/safe_dump_spa_FNM00131203264_2013-10-11_17_14_08_8666_safe’,file type elf64-x86-64.

[0052] 0x0000000000400000-0x0000000000404000is load1

[0053] 0x0000000000603000-0x0000000000604000is load2

[0054] …

[0055] 0x00007f6f74a111a8-0x00007f6f74a13cb8is.plt in/tmp/abcdt/a/564217-spa_FNM00131203264_2013-10-11_17_14_08_8666/EMC/csx/ulib64/csx_urt.so

[0056] 0x00007f6f74a13cc0-0x00007f6f74b5bab8is.text in/tmp/abcdt/a/564217-spa_FNM00131203264_2013-10-11_17_14_08_8666/EMC/csx/ulib64/csx_urt.so

[0057] 0x00007f6f74b5bac0-0x00007f6f74b5bdf1is csx_gx_rt in/tmp/abcdt/a/564217-spa_FNM00131203264_2013-10-11_17_14_08_8666/EMC/csx/ulib64/csx_urt.so

[0058] …

[0059] 在这个示例性实施例中，经过上面的查询过程，能够得出与故障相关的函数的模块名称是csx_urt.so，该模块名称信息可以与函数名称信息csx_rt_proc_do_abort一起来形成相应的简化堆栈帧。

[0060] 为了更好地区分出函数名称信息和模块名称信息，在一些实施例中，在相应的简化堆栈帧中，在相应的函数名称信息与相应的模块名称信息之间可以设置分隔符。例如，在一种具体的实现中，可以使用符号“!”来作为分隔符。本领域的技术人员可以理解，其他任何不会引起混淆的符号都可以被使用作为相应的函数名称信息与相应的模块名称信息之间的分隔符。

[0061] 在使用“!”作为分隔符的实施例中，通过利用函数名称信息和模块名称信息，图1中所示出的编号为#2的堆栈帧能够被简化成为如下的形式：csx_urt.so!csx_rt_proc_do_abort。本领域的技术人员可以理解，该具体形式仅为一种示例，本领域的技术人员可以通过调整信息的前后顺序、使用不同的分隔符、或者添加其他的可变或者固定信息来形成各种不同的简化堆栈帧。

[0062] 在一个实施例中，针对图1中所示出的堆栈进行方法200的步骤201和202之后，可以得出下面所示出的简化版本的新堆栈文本。它清楚地示出了故障转储的堆栈中所涉及到的函数和模块。另外，从实践的角度来看，方法200的步骤201-202能够通过一些自动化工具/脚本来进行。

[0063] libc.so.6!raise

[0064] libc.so.6!abort

[0065] csx_urt.so!csx_rt_proc_do_abort

[0066] csx_urt.so!csx_rt_assert_int_take_user_space_panic_action

[0067] csx_urt.so!csx_rt_assert_request_panic_with_info

[0068] EmcPAL.sys!EmcpalBugCheck

[0069] ……

[0070] csx_urt.so!csx_rt_cpi_thread_command

[0071] csx_urt.so!csx_rt_sked_thread_wrapper

[0072] libpthread.so.0!start_thread

[0073] libc.so.6!clone

[0074] 接着，方法200可以进入步骤203。在步骤203中，方法200的执行主体可以基于故障的简化堆栈之间的相似度来确定故障之间的相似度。

[0075] 本领域的技术人员可以理解，故障之间的相似度可以通过故障的转储堆栈之间的相似度来体现，而故障转储的堆栈包括多个堆栈帧。这些堆栈帧经过简化之后，已经去除了总是变化的噪声信息，而保留了更为关键的函数名称信息和/或模块名称信息。因此，在一些实施例中，可以通过直接比较两个故障的简化堆栈之间的相似简化堆栈帧的数目，来确定故障之间的相似度。

[0076] 此外，本领域的技术人员可以意识到，故障转储的堆栈中的堆栈帧具有时间上的先后顺序，因此在确定故障之间的相似度时将堆栈帧之间的先后关系纳入考虑能够更好地体现两次故障之间的相似度。

[0077] 因此，在一些实施例中，可以从简化堆栈底部的简化堆栈帧开始向顶部方向逐个地移除简化堆栈帧直到仅剩余顶部的简化堆栈帧，每移除一个简化堆栈帧生成一个子堆栈；并且基于故障的简化堆栈及其子堆栈之间的相似度来确定故障之间的相似度。

[0078] 这样做的益处在于，这些子堆栈可以保持堆栈中的次序信息，因为调用次序是导

致转储的代码路径的关键分支。如果有问题的代码的某个部分能够被来自相同/不同模块的不同上层代码调用，则从问题代码之后，它们将很可能导致具有完全相同的子堆栈的相似堆栈。例如，上面所示出的经简化的堆栈能够具有以下的子堆栈。

[0079] 子堆栈-0(SubStack-0):<即为简化堆栈本身>

[0080] libc.so.6!raise

[0081] libc.so.6!abort

[0082] csx_urt.so!csx_rt_proc_do_abort

[0083] csx_urt.so!csx_rt_assert_int_take_user_space_panic_action

[0084] csx_urt.so!csx_rt_assert_request_panic_with_info

[0085] EmcPAL.sys!EmcpalBugCheck

[0086] ……

[0087] csx_urt.so!csx_rt_cpi_thread_command

[0088] csx_urt.so!csx_rt_sked_thread_wrapper

[0089] libpthread.so.0!start_thread

[0090] libc.so.6!clone

[0091] 子堆栈-1(SubStack-1):

[0092] libc.so.6!raise

[0093] libc.so.6!abort

[0094] csx_urt.so!csx_rt_proc_do_abort

[0095] csx_urt.so!csx_rt_assert_int_take_user_space_panic_action

[0096] csx_urt.so!csx_rt_assert_request_panic_with_info

[0097] EmcPAL.sys!EmcpalBugCheck

[0098] ……

[0099] csx_urt.so!csx_rt_cpi_thread_command

[0100] csx_urt.so!csx_rt_sked_thread_wrapper

[0101] libpthread.so.0!start_thread

[0102] ~~libc.so.6!clone~~

[0103] 子堆栈-2(SubStack-2):

[0104] libc.so.6!raise

[0105] libc.so.6!abort

[0106] csx_urt.so!csx_rt_proc_do_abort

[0107] csx_urt.so!csx_rt_assert_int_take_user_space_panic_action

[0108] csx_urt.so!csx_rt_assert_request_panic_with_info

[0109] EmcPAL.sys!EmcpalBugCheck

[0110] ……

[0111] csx_urt.so!csx_rt_cpi_thread_command

[0112] csx_urt.so!csx_rt_sked_thread_wrapper

[0113] ~~libpthread.so.0!start_thread~~

[0114] ~~libc.so.6!clone~~

[0115] …

[0116] …

[0117] 子堆栈-29(Sub stack-29):

[0118] libc.so.6!raise

[0119] libc.so.6!abort

[0120] csx_urt.so!csx_rt_proc_do_abort

[0121] 子堆栈-30(SubStack-30):

[0122] libc.so.6!raise

[0123] libc.so.6!abort

[0124] 子堆栈-31(SubStack-31):

[0125] libc.so.6!raise

[0126] 在一些特定的实施例中,两个故障之间的相似度可以由它们的故障转储堆栈共有的具有最多堆栈帧的子堆栈中的堆栈帧数目来确定。在另一种情况中,有可能两个故障之间具有完全相同的简化堆栈帧。因此,可以基于故障之间所共有的简化堆栈或者具有最多简化堆栈帧的子堆栈来确定故障之间的相似度。

[0127] 在一些实施例中,可以针对多次故障中的每次故障来获得相应的简化堆栈及其子堆栈,并且形成多次故障的所有简化堆栈和所有子堆栈与多次故障之间的对应关系表格。

[0128] 本领域的技术人员可以理解,在形成了这样的对应关系表格之后,就可以方便地通过查询该表格来确定子堆栈与故障(即故障转储)之间的对应关系,从而可以快速地确定任意两个故障之间具有多少相同或相似的子堆栈,并且可以确定任意两个故障所共有的具有最多堆栈帧的子堆栈,从而确定相似度。在一些特定的实施例中,可以使用哈希算法来形成该对应关系表格,以便于针对数量庞大的故障转储快速地形成该对应关系表格。

[0129] 因此,在一些实施例中,可以针对新故障来获得相应的简化堆栈及其子堆栈,并且在对应关系表格中查找新故障的简化堆栈及其子堆栈,以确定与新故障具有相同简化堆栈或子堆栈的故障。

[0130] 此外,可以将新故障的不在对应关系表格中的简化堆栈和子堆栈添加到对应关系表格中。因此,该对应关系表格可以随着故障数目的增长而不断被扩充,从而后来发生的故障找到先前发生的相似故障的可能性将会增加。

[0131] 在一种具体的实施方式中,可以首先将简化堆栈分解为子堆栈,并且可以利用列表或数组的哈希对所准备的堆栈及其子堆栈进行哈希。此外,可以将故障转储的标记附加到具有相同堆栈或子堆栈的哈希列表或数组。故障转储标记可以是具有某个时间戳的故障转储名称或者用以唯一识别故障转储的序列号信息。

[0132] 然后,可以对所有故障转储的子堆栈进行哈希以将所有的故障转储基于它们的故障转储标记归组为哈希表格。这仅需要在系统首次启动时进行一次,在此之后仅需要将新产生的故障转储添加到哈希表格中。该过程可以通过Perl语言表示如下。

[0133] %Stack_hash;

[0134] $Stack_Hash{$each_sub_stack}＝[dump_1,dump_2,…]

[0135] 通过循环经过所有的转储,对于具有子堆栈0到子堆栈N的特定转储而言的示例性哈希列表能够被描述如下。

[0136]

子堆栈	故障转储（dmp）列表					
SubStack-0	dmp1	dmp2				
SubStack-1	dmp1	dmp2	dmp3			
SubStack-2	dmp1	dmp2	dmp3	dmp4	dmp5	

[0137]

…	…	…				
SubStack-N	dmp1	dmp2	dmp3	dmp4	dmp5	…

[0138] 如该表格所示出的，子堆栈中的堆栈数目越少，越多的各种转储将会与之相联系。因此，如果不能找到完全相同的堆栈，则可以找到部分相同(即，具有相同的子堆栈)的堆栈，其相比于完全堆栈而言具有共有相同的部分代码路径的较少数目的堆栈。

[0139] 此外，从工程上的角度来考虑，该查询过程可以停止在可能认为已经足够的某些点处。例如，可以停止在大多数时间共享相同的代码路径的某个已知的函数处。

[0140] 一旦生成了新的转储，可以首先在哈希表格中查找其简化的堆栈，以查看是否任何其他的(多个)转储共有完全相同的简化堆栈。如果没有，则可以从简化堆栈的顶部开始查找它的子堆栈。如此，最佳匹配堆栈能够由具有相匹配的具有最多数目堆栈子帧的子堆栈来定义，并且还能够以这种方式选出排序靠前的相似故障转储。

[0141] 通过将子堆栈视为具有次序上下文的不可分的块，可以对所选择的相似堆栈的相似度具有更好的体现。此外，通过所附加的模块名称信息，可以识别所牵涉的模块并且可以为自动分诊定义分诊规则。

[0142] 随着子堆栈具有更多数目的匹配堆栈帧，则它们具有更多相同的代码路径，即共享转储堆栈之间的更多相似性。因此，匹配的代码路径的比率可以用来量化该相似性。此外，这些哈希列表结果能够容易地被保存并且重新加载到存储器中以用于快速查找，以便确定是否任何新生成的转储具有准确相同或者相似的堆栈。

[0143] 图3示意性地示出了根据本公开内容的实施例的装置300的框图。在一些实施例中，装置300可以用于计算设备，该计算设备在每次发生故障时可以生成用于故障转储的堆栈，每个堆栈从底部到顶部可以包括多个堆栈帧，每个堆栈帧可以包括与故障有关的函数信息。在图3中，使用虚线框来表示可选的单元。

[0144] 本领域的技术人员可以理解，图3中仅示出了装置300中的与本公开的实施例紧密相关的单元或组件，在具体的实践中，装置300可以包括使其能够正常操作的其他功能单元或组件。此外，本领域的技术人员还可以理解，装置300的各个单元之间可以存在必要的连接。

[0145] 如图3中所示出的，装置300可以包括提取单元301、简化单元302和相似度确定单元303。在一些实施例中，提取单元301可以被配置为从所述堆栈中的每个堆栈帧中提取相应的函数名称信息；简化单元302可以被配置为基于每个堆栈帧的相应的函数名称信息来形成相应的简化堆栈帧，以获得堆栈的简化堆栈；以及相似度确定单元303可以被配置为基于故障的简化堆栈之间的相似度来确定故障之间的相似度。

[0146] 在一些实施例中，简化单元302可以进一步被配置为仅利用每个堆栈帧的相应的函数名称信息来形成相应的简化堆栈帧。在一些实施例中，简化单元302可以进一步被配置为利用每个堆栈帧的相应的函数名称信息和相应的模块名称信息来形成相应的简化堆栈帧。

[0147] 在一些实施例中，简化单元302可以进一步被配置为从每个堆栈帧中提取相应的模块名称信息，或者通过查找每个堆栈帧中的函数地址信息来确定相应的模块名称信息。在一些实施例中，简化单元302可以进一步被配置为，在相应的简化堆栈帧中，在所述相应的函数名称信息与所述相应的模块名称信息之间设置分隔符。

[0148] 在一些实施例中，装置300可以进一步包括子堆栈生成单元304。子堆栈生成单元304可以被配置为从简化堆栈底部的简化堆栈帧开始向顶部方向逐个地移除简化堆栈帧直到仅剩余顶部的简化堆栈帧，每移除一个简化堆栈帧生成一个子堆栈。在这些实施例中，相似度确定单元303可以进一步被配置为基于故障的简化堆栈及其子堆栈之间的相似度来确定故障之间的相似度。

[0149] 在一些实施例中，简化单元302可以进一步被配置为基于故障之间所共有的简化堆栈或者具有最多简化堆栈帧的子堆栈来确定故障之间的相似度。

[0150] 在一些实施例中，简化单元302可以进一步被配置为针对多次故障中的每次故障来获得相应的简化堆栈及其子堆栈。在这些实施例中，装置300可以进一步包括对应关系表格形成单元305。对应关系表格形成单元305可以被配置为形成多次故障的所有简化堆栈和所有子堆栈与多次故障之间的对应关系表格。

[0151] 在一些实施例中，简化单元302可以进一步被配置为针对新故障来获得相应的简化堆栈及其子堆栈。在这些实施例中，相似度确定单元303可以进一步被配置为在对应关系表格中查找新故障的简化堆栈及其子堆栈，以确定与新故障具有相同简化堆栈或子堆栈的故障。

[0152] 在一些实施例中，装置300可以进一步包括添加单元306。添加单元306可以被配置为将新故障的不在对应关系表格中的简化堆栈和子堆栈添加到对应关系表格中。

[0153] 在一些实施例中，对应关系表格形成单元305可以进一步被配置为使用哈希算法来形成对应关系表格。

[0154] 此外，本领域的技术人员可以理解，由于本公开的实施例的方法实际上利用了故障转储堆栈包括代码路径的严格的依次序列，所以本公开内容的实施例所提出的方法也能够扩展到其他具有严格依次顺序的系列事件中，以快速地找到一连串事件是否类似于另一连串事件。广义地进行理解，本公开内容的实施例中的每个堆栈帧即可以被视为是一个事件。

[0155] 图4示意性地示出了可以用来实现本公开内容的实施例的示例性计算机系统/服务器412的框图。应当注意，图4中所示出的计算机系统/服务器412仅是一种示例，不对本公开内容的实施方式的功能和使用范围进行任何限制。

[0156] 如图4中所示出的，计算机系统/服务器412以通用计算设备的形式表现。计算机系统/服务器412的组件可以包括但不限于：一个或者多个处理器或者处理单元416，系统存储器428，连接不同系统组件(包括系统存储器428和处理单元416)的总线418。

[0157] 总线418表示几类总线结构中的一种或多种，包括存储器总线或者存储器控制器，外围总线，图形加速端口，处理器或者使用多种总线结构中的任意总线结构的局域总线。举例来说，这些体系结构包括但不限于工业标准体系结构(ISA)总线，微通道体系结构(MAC)总线，增强型ISA总线、视频电子标准协会(VESA)局域总线以及外围组件互连(PCI)总线。

[0158] 计算机系统/服务器412典型地包括多种计算机系统可读介质。这些介质可以是任

何能够被计算机系统/服务器412访问的可用介质，包括易失性和非易失性介质，可移除的和不可移除的介质。

[0159] 系统存储器428可以包括易失性存储器形式的计算机系统可读介质，例如，存储器430和/或缓存器432。计算机系统/服务器412可以进一步包括其他可移除/不可移除的、易失性/非易失性计算机系统存储介质。尽管图4中未示出，但是可以提供用于对可移除非易失性磁盘(例如“软盘”)读写的磁盘，以及对可移除非易失性光盘(例如CD-ROM、DVD-ROM或者其他光介质)读写的光盘。在这些情况下，每个磁盘可以通过一个或者多个数据介质接口与总线418相连。存储器428可以包括至少一个程序产品，该程序产品具有一组(例如至少一个)程序模块，这些程序模块被配置以执行本公开内容的各实施方式的功能。

[0160] 具有至少一个程序模块442的程序/实用工具440，可以存储在例如存储器428中，这样的程序模块442包括但不限于：操作系统、一个或者多个应用程序、其他程序模块以及程序数据，这些示例中的每一个或某种组合中可能包括网络环境的实现。程序模块442通常执行本公开内容所描述的实施方式中的功能和/或方法。

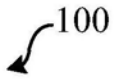
[0161] 根据需要，计算机系统/服务器412也可以与一个或多个外部设备(例如显示设备424、存储设备414等)通信，还可与一个或者多个使得用户能与该计算机系统/服务器412交互的设备通信，和/或与使得该计算机系统/服务器412能与一个或多个其他计算设备进行通信的任何设备(例如网卡，调制解调器等等)通信。这种通信可以通过输入/输出(I/O)接口422进行。并且，计算机系统/服务器412还可以通过网络适配器420与一个或者多个网络(例如局域网(LAN)，广域网(WAN)和/或公共网络，例如因特网)通信。如图所示，网络适配器420通过总线418与计算机系统/服务器412的其他模块通信。应当明白，尽管图中未示出，可以结合计算机系统/服务器412使用其他硬件和/或软件模块，包括但不限于：微代码、设备磁盘、冗余处理单元、外部磁盘驱动阵列、RAID系统、磁带磁盘以及数据备份存储系统等。

[0162] 在对本公开内容的实施例的描述中，术语“包括”及其类似用语应当理解为开放性包含，即“包括但不限于”。术语“基于”应当理解为“至少部分地基于”。术语“一个实施例”或“该实施例”应当理解为“至少一个实施例”。

[0163] 应当注意，本公开内容的实施例可以通过硬件、软件或者软件和硬件的结合来实现。硬件部分可以利用专用逻辑来实现；软件部分可以存储在存储器中，由适当的指令执行系统，例如微处理器或者专用设计硬件来执行。本领域的技术人员可以理解上述的设备和方法可以使用计算机可执行指令和/或包含在处理器控制代码中来实现，例如在可编程的存储器或者诸如光学或电子信号载体的数据载体上提供了这样的代码。

[0164] 此外，尽管在附图中以特定顺序描述了本公开内容的方法的操作，但是这并非要求或者暗示必须按照该特定顺序来执行这些操作，或是必须执行全部所示的操作才能实现期望的结果。相反，流程图中描绘的步骤可以改变执行顺序。附加地或备选地，可以省略某些步骤，将多个步骤组合为一个步骤执行，和/或将一个步骤分解为多个步骤执行。还应当注意，根据本公开内容的两个或更多装置的特征和功能可以在一个装置中具体化。反之，上文描述的一个装置的特征和功能可以进一步划分为由多个装置来具体化。

[0165] 虽然已经参考若干具体实施例描述了本公开内容，但是应当理解，本公开内容不限于所公开的具体实施例。本公开内容旨在涵盖所附权利要求的精神和范围内所包括的各种修改和等效布置。




```
(gdb)bt
#0  0x00007f6f735e3b35 in raise ()
from/tmp/abcdt/a/564217-spa_FNM00131203264_2013-10-11_17_14_08_8666/lib64/libc.so.6
#1  0x00007f6f735e5111 in abort ()
from/tmp/abcdt/a/564217-spa_FNM00131203264_2013-10-11_17_14_08_8666/lib64/libc.so.6
#2  0x00007f6f74ac04d7in csx_rt_proc_do_abort ()
at ../../../../../../csx/csxroot/src/csx_infra/csx_rt/csx_rt_osll/csx_rt_proc.c:2450
#3  0x00007f6f74a57966 in csx_rt_assert_int_take_user_space_panic_action (
panic_action=1)
at ../../../../../../csx/csxroot/src/csx_infra/csx_rt/csx_rt_misc/csx_rt_assert.c:786
#4  0x00007f6f74a57d13 in csx_rt_assert_request_panic_with_info (
failure_state_info=0x1,
file=0x7f6f7083d6b0"/c4_working/Coding/KH_sandbox-kpi/safe/catmerge/EmcPAL/Driver/EmcPAL_Misc.c", line=73, argcount=6,
format=0x7f6f7083d678 "bugcheck: %08x [%08llx] [%08llx] [%08llx] [%08llx]\n")
at ../../../../../../csx/csxroot/src/csx_infra/csx_rt/csx_rt_misc/csx_rt_assert.c:960
#5  0x00007f6f70835502 in EmcpalBugCheck (BugCheckCode=0,
    BugCheckParameter1=0, BugCheckParameter2=100663296, BugCheckParameter3=1,
    BugCheckParameter4=0)
at /c4_working/Coding/KH_sandbox-kpi/safe/catmerge/EmcPAL/Driver/EmcPAL_Misc.c:68
  ......
  #28 0x00007f6f74a6732c in csx_rt_cpi_thread_command (thr_context=0xf380fc0)
  at ../../../../../../csx/csxroot/src/csx_infra/csx_rt/csx_rt_cp/csx_rt_cpi.c:178
  #29 0x00007f6f74a2a7f4 in csx_rt_sked_thread_wrapper (context=0xf381090)
  at ../../../../../../csx/csxroot/src/csx_infra/csx_rt/csx_rt_sked/csx_rt_sked_llp_uu.c:914
  #30 0x00007f6f73e43d96 in start_thread ()
  from /tmp/abcdt/a/564217-spa_FNM00131203264_2013-10-11_17_14_08_8666/lib64/libpthread.so.0
  #31 0x00007f6f7368ac5d in clone ()
  from /tmp/abcdt/a/564217-spa_FNM00131203264_2013-10-11_17_14_08_8666/lib64/libc.so.6
```

图1

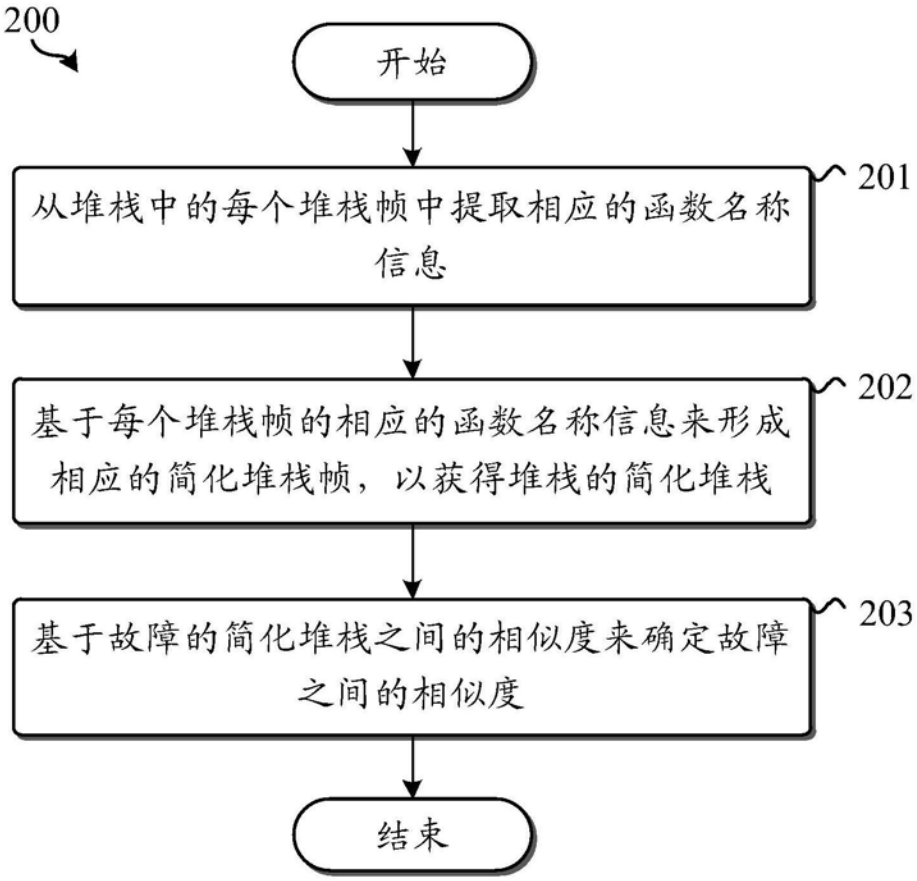

200
开始
从堆栈中的每个堆栈帧中提取相应的函数名称信息
201
基于每个堆栈帧的相应的函数名称信息来形成相应的简化堆栈帧，以获得堆栈的简化堆栈
202
基于故障的简化堆栈之间的相似度来确定故障之间的相似度
203
结束

图2

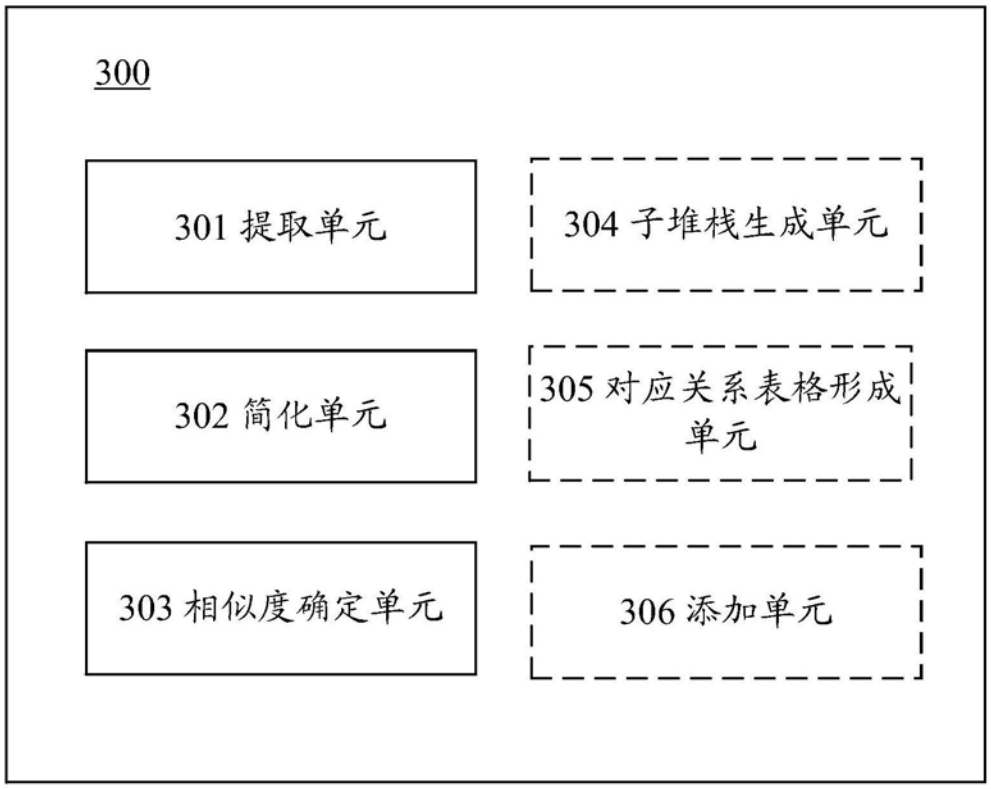

300
301 提取单元
304 子堆栈生成单元
302 简化单元
305 对应关系表格形成单元
303 相似度确定单元
306 添加单元

图3

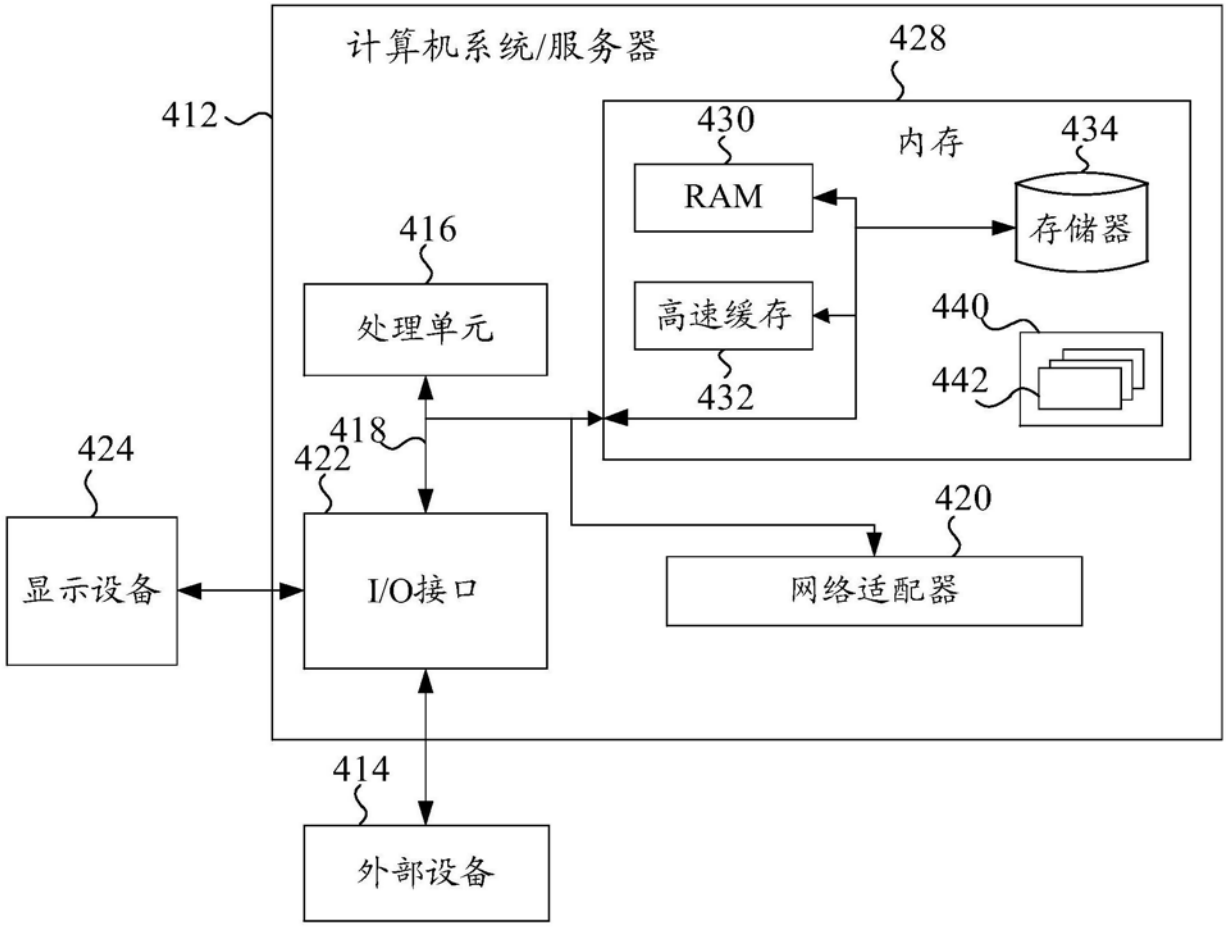

计算机系统/服务器
428
412
430
内存
434
RAM
存储器
416
处理单元
高速缓存
440
442
432
418
424
422
420
显示设备
I/O接口
网络适配器
414
外部设备

图4