基于日志分析的虚拟化环境故障定位系统①

田 斐[1,2], 吴 恒[1,2], 张文博[1]

[1](中国科学院软件研究所 软件工程技术研究开发中心, 北京 100190)
[2](中国科学院大学, 北京 100049)

摘 要: 在云平台的运行维护中, 虚拟化环境的故障定位对容错极其重要. 针对虚拟化环境的空间拓扑时变性, 提出了一种基于时变的节点间依赖关系的日志分析方法, 为决策树故障定位提供支持. 节点间依赖关系能够帮助压缩日志, 只提供故障时间窗口内的有效信息作为挖掘来源, 从而更快更准确地完成故障定位. 实验表明, 能够在较大系统规模的多变的虚拟化环境下实现有效和高效的故障定位.

关键词: 故障定位; 决策树; 虚拟化; 日志压缩; 节点间依赖关系

Virtualization Environment Fault Localization System Based Log Analyzing

TIAN Fei[1,2], WU Heng[1,2], ZHANG Wen-Bo[2]

[1](Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: Fault localization is extremely important for fault tolerance in Virtualization-based cloud platform. This paper proposes an inter-node dependency-based log analysis approach consider the time-varying spatial topological using decision tree. Dependencies between nodes can help compressing log, providing valid information for analysing, thereby completing the fault localization faster and more accurately. Experiment results show that our approach can help to achieve effective and efficient fault localization in the larger-scale virtual environment variable.

Key words: fault localization; decision tree; virtualization environment; log compression; inter-node dependency

1 背景

随着信息化技术的发展, 云计算已经成为构建新型信息化系统的主流计算泛型. 根据美国国家标准与技术研究院[1](NIST)的定义, 云计算是一种利用互联网实现随时随地、按需、便捷地访问共享资源池(如计算设施, 存储设备, 应用程序等)的计算模式.

容错是云计算的重要特征, 即系统能够在不断发生故障的环境中不间断运行, 因此故障管理是云平台运行维护的重要组成部分. 故障管理通常包括 3 个步骤: 探测系统的运行时故障, 定位故障的原因, 根据定位结果采取相应的容错操作. 在上述过程中, 故障定位基于故障探测结果, 为后续容错操作提供触发条件, 定位的准确性对于系统的运行维护非要重要.

在故障定位方法上, 由于系统日志是反映系统行为的天然数据, 而日志信息又是各系统的标准配置, 相对于专门搭建监测系统, 基于日志的分析有丰富的数据源并具备通用性. 因此大量的相关工作以日志信息为研究对象, 通过日志分析进行系统故障定位[2-5], 其核心思想是采用故障注入的方法对业务系统正常行为或者异常行为进行建模, 通过运行时日志分析技术进行故障定位. 系统日志作为系统运行时状态最常用的记录, 为故障定位提供了丰富的信息, 然而通常情况下, 系统日志由于包含了大量的冗余信息, 无法直接用作数据分析和建模, 因此为更准确高效地进行故障定位而进行日志数据预处理成为了必然. 当前的预处理技术主要以高压缩比为目的, 普遍在时间维度行过滤[5,6], 对指定时间窗口的重复错误日志进行删除. 工作[5]指出这样的压缩方式会漏掉那些特征为一长串

① 基金项目:国家自然科学基金(61173003,61363003)
收稿时间:2014-03-09;收到修改稿时间:2014-03-31

警告的故障类型, 影响到故障定位的精确度.

日志高压缩比和检验精确度之间平衡是日志分析的关键技术之一. 而虚拟化作为构建云计算关键技术之一, 在提高业务系统灵活性的同时, 也给日志压缩带来了极大的挑战[7]. 仅仅以时间维度进行日志过滤的方法无法适应复杂多变的虚拟化环境, 因为虚拟化环境具有空间拓扑时变性的新特征. 例如虚拟机迁移, 虚拟机迁移前后的日志信息存放在不同的宿主物理机上, 只有合并迁移前后的日志才能准确反映虚拟机错误执行的完整状态. 如果仅仅依据时间维度进行日志过滤, 移除迁移后的宿主物理机上该虚拟机的错误日志, 则会丢失当前虚拟机的位置信息; 若移除迁移前的宿主物理机上该虚拟机的错误日志, 则漏掉了那些特征为一长串警告的故障类型, 都可能导致无法准确定位的结果. 总的来说, 不同宿主物理机的日志可能共同构成某故障的特征类型, 而这些宿主物理机之间的依赖关系又可能随着时间发生改变.

本文将云平台中宿主物理机之间的随着时间变化的依赖关系作为日志压缩预处理和故障特征建模及定位的依据的一部分, 从虚拟机为主体的错误日志入手, 以时间窗口内该虚拟机所有相关宿主物理机的日志为分析来源, 进行故障特征建模. 这样过滤掉了那些无关物理机上的日志信息, 同时也充分保留了时间窗口内的有效信息. 由于宿主物理机之间的依赖关系需要由虚拟机来进行关联, 所以进行依赖关系分析时需要将虚拟机和所有物理机上的虚拟化组件列入考虑, 以下将虚拟机、计算虚拟化组件、存储虚拟化组件和网络虚拟化组件统称为节点.

本文针对虚拟化环境的空间拓扑时变性, 提出了一种基于时变的节点间依赖关系的日志压缩方法, 从而为之后基于决策树的故障定位方法提供了良好的支持. 通过在 OnceCloud 系统上对于精确度和召回率的检验, 以及对比是否使用日志过滤下故障定位时间随系统规模的变化, 本文提出的方法能够在较大系统规模的多变的虚拟化环境下快速诊断故障.

论文的后续组织形式如下: 第二节介绍基于日志的故障定位的相关工作, 第三节给出了故障定位的技术原理, 第四节描述了系统的设计与实现, 第五节为实验与分析, 第六节为总结.

2 相关工作

基于日志进行故障定位的相关工作有很多, 大致可以按是否需要故障重放分为两类, 一类是不需要故障重放的建模方法, 使用统计分析方法来建模系统的正常行为模型, 或者假设可以获得系统源代码[4,8], 崩溃后根据系统遗留下来的日志, 在源代码中进行查找匹配, 找出系统崩溃之前系统执行的路径并分析其中的函数调用关系从而帮助程序员进行故障定位. 虽然这种方法能够获得较为细粒度的故障检测结果, 但源代码并非所有系统都能获得, 且停机调试对于系统的正常运行会是很大的干扰. 另一类则是需要故障重放来进行建模, 采用故障注入的方法对业务系统正常行为或者异常行为进行建模, 通过运行时日志分析技术进行故障定位[2,3,5].

过滤冗余日志和去除噪声日志也一直是研究焦点. 当前工作普遍在时间维度上进行过滤[6] , 主要以高压缩比为目的, 甚至能够达到 99.96%的压缩比. 文献[5]指出这样的压缩方式会漏掉某些故障类型, 从而影响故障定位的精确度, 并提出了一种相对保证精确度的前提下有效过滤冗余日志的方法. 文献[2]提出了一种基于相似度的过滤方法, 通过比较故障注入之前的日志和注入之后的日志的相似度来过滤噪声日志, 提高模型有效性.

本文针对虚拟化环境中空间拓扑随时间变化的特性, 提出了一种基于节点间依赖关系的日志分析方法, 为基于决策树的故障定位提供支持. 把节点依赖关系作为输入数据能够帮助有效的压缩日志, 更快更准确地完成故障定位.

3 技术原理

本文主要研究云平台上虚拟化环境中基于日志分析的故障定位方法, 以设计一个高效可用的故障定位系统为目标. 本文采用的思路为维护时间窗口内节点间依赖关系信息, 在计算节点日志中检测到故障日志时根据依赖关系找出相关节点, 将这些节点的日志合并建模, 并使用分类器进行故障定位. 下面的一至三小节分别阐述了根据日志训练模型, 建立依赖关系和合并日志三个阶段的关键方法.

3.1 基于决策树模型的特征建模

为了识别整个故障发生过程中出现的特征, 针对系统日志进行建模, 构建特征识别器. 特征识别器实际上是一个分类器, 将日志中体现的特征映射为预设的故障类型. 常见的分类器构造方法包括决策树分类

法、神经网络、贝叶斯分类法等. 基本思路是选择一种学习算法用来确定分类模型, 从而拟合样本数据中故障类型和日志属性集之间的映射关系, 这样面对未知故障类型的样本数据时, 可以基于此分类模型预测样本的故障类型.

在构建分类器的过程中, 需要将数据划分为训练集和检验集, 训练集是标识了故障类型的数据, 用来建模; 检验集未标识故障类型, 用来检验模型的有效性. 有效性可用精确率和召回率进行衡量.

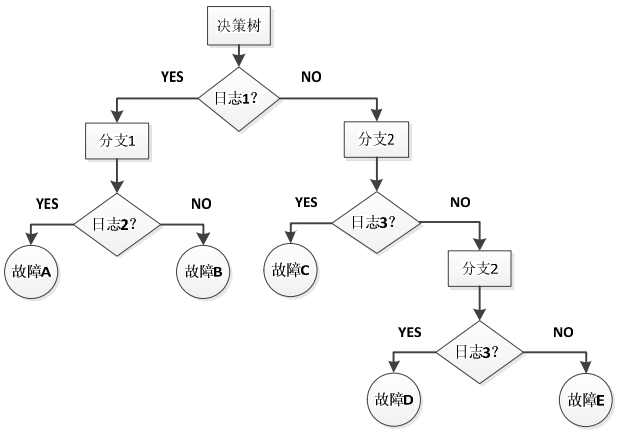


图 1　决策树

决策树是一种简单有效的分类技术, 通常用于处理多属性数据的分类问题, 在处理离散字段上很有效. 为了提取故障特征, 我们使用故障发生时间窗口内日志特征向量作为基本分析单元. 假设虚拟化环境内共存在有 m 类日志, 则一个故障特征拥有 m 维属性, 每一维属性的取值为{0, 1}或者{NO, YES}, 0 或 NO 代表该类日志未出现, 1 或 yes 代表该类日志出现, 这样用来描述该类日志的出现与否与该故障发生的关联性.

选用 Weka 的 J48 决策树模块作为我们提取故障特征的工具. J48 是 C4．5 决策树的著名实现.

3.2 基于有向无环图的节点依赖关系刻画

有向无环图 DAG(Directed Acrylic Graph)常常用来刻画节点间依赖关系, 通常用用图中点间的边来表示节点之间的依赖. 本文采用加权有向无环图, 节点分为虚拟节点和物理节点, 图中用圆形表示虚拟节点, 方形表示物理节点, 物理节点分为计算节点(安装 Xen[9]或 KVM[10]等计算虚拟化软件), 存储节点(磁盘阵列, 远程存储器等)和虚拟网络节点. 虚拟节点一般来说代表虚拟机. 节点或边的权值 α 表示该对象在 α 秒之前被删除, 若 α=0, 则表示该对象在系统中依然存在. DAG 图有重要参数时间窗口 w, 与故障检验窗口相等, DAG 图最多保存 2w 秒之前的历史节点依赖关系, 有 α<2w.

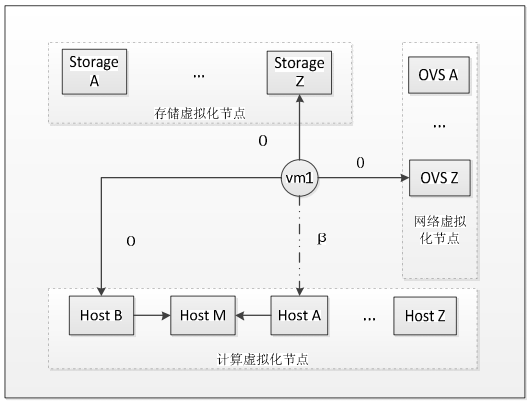


图 2　有向无环图示例

以图 2 为例, Host A, Host B 和 Host M 组成一个计算节点资源池, Host M 是资源池主节点, Host A 和 Host B 是从节点, 从节点由主节点进行管理, 因此主节点的状态也会影响从节点. Storage A 到 Storage Z 为独立的存储节点. OVS A 到 OVS Z 为独立的虚拟网络. Vm1 的磁盘文件存放在 Storage A 上, 使用 OVS Z 来上网. 在 β 秒前, vm1 从 Host A 迁移到了 Host B 上. 图 3 中节点的权值均为 0, 边的权值有 vm1->Host A 权值为 β, 其他边权值也均为 0.

描述节点间依赖关系的 DAG 可以在构建云平台的过程中进行构建, 并随着云平台拓扑结构的变化而不断发生改变.

3.3 相关日志的合并

从第一节给出的示例日志可以看出, 一条错误日志通常包括 4 个字段, 分别为日志产生时间, 日志严重级别, 产生日志的代码行和日志描述信息. 其中若日志严重级别为 error 或 fatal, 日志描述信息中会记录产生错误的对象标识. 同时, 根据日志发生的节点, 我们还可以加上节点的 ip 信息和节点的类型. 格式如下:

(Time) Level (Path) Description(Node UUID)

为了降低对系统的负担, 只考虑 error 或 fatal 级别的日志. 通过找出出错节点, 在节点依赖关系图中找出以出错节点为根的子图, 则图中所有节点皆为故障相关节点. 找出故障时间窗口内相关节点的所有日志并进行合并, 这些日志成为了构建决策树的基础.

需要指出的是，在分布式系统中，不同机器之间需要进行时钟同步. 系统中采用 NTP(network time protocol)服务器进行时钟同步，可以做到在目标系统中将时钟误差限制在 1s 内.

4 系统设计与实现

上一节介绍了本文的主要思路和方法，这一节将从系统的总体框架、关键模块设计与实现等方面来具体介绍系统的设计与实现.

4.1 故障定位系统框架

在模型训练阶段和检验阶段的过程基本上是一致的. 模型提取阶段的流程如下:

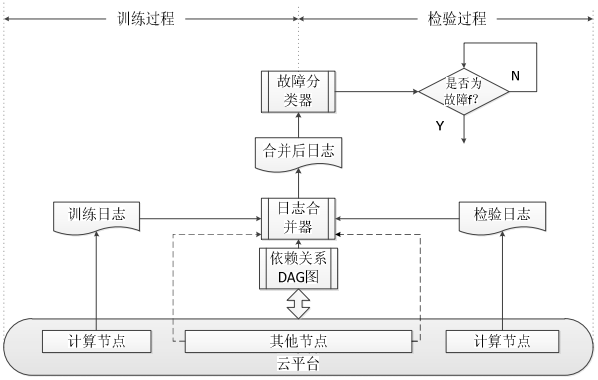


图 3 故障定位系统框架

(1)从计算节点中提取出错日志(标记为 error 和 fatal 的日志);

(2)节点依赖关系图用需随云平台空间拓扑状态的变化而更新，并至少保存当前时间之前两个时间窗口内的历史数据. 根据节点依赖关系图，找出在故障发生窗口内与出错节点有依赖关系的节点，从这些节点中提取故障发生窗口内的日志，进行合并;

(3)对于合并好的日志，使用 C4.5 决策树模型构造一个基于规则的故障分类器.

与此相应，运行时故障识别过程如下:

(1)-(2)同模型提取阶段

(3)通过故障分类器识别合并好的日志的故障类型.

云平台在运行过程中空间拓扑不断发生改变，节点依赖关系 DAG 图反映了故障时间窗口内云平台空间拓扑状态，会随着空间拓扑状态的变化而不断更新.

4.2 模块设计

在整个框架中，有三大重要模块，分别是为日志过滤提供依据的节点依赖关系维护模块，负责将相关日志信息从不同节点收集并合并的日志合并模块，以及基于决策树的故障分类器. 下面将详细阐述这些模块所涉及的设计要点.

4.2.1 节点依赖关系维护模块

该模块负责维护节点间依赖关系，需为日志合并模块提供查询接口，需随云平台空间拓扑状态的变化而更新，为了能够充分体现时间窗口内空间拓扑的状态，会保存当前时间之前两个时间窗口内的历史信息.

依赖关系用一个有向无环图来表示，DAG 图可以表现为一个三元组 G=<V, E, w>，其中，V 是节点的集合{v1,v2,…vN}，E 是节点间的边的集合{e1,e2,...eN}，w 为故障检验时间窗口. 其中 vi=<node i,α>, ei=<vj,vk,β>. node i 代表云平台中虚拟节点或物理节点，即虚拟机，计算节点，存储节点或网络节点，α 代表 vi 的权值. ei 代表 vj 和 vk 之间存在 vj 到 vk 的依赖关系，权值为 β. 其中 α<2w, β<2w. 在实际实现中，使用 v2eMap<v,eList>来存储有向无环图.

本模块提供两大接口:

① 更新(update): 根据云平台拓扑结构的变化进行更新. 输入为更改云平台结构的操作，输出为更新后的 DAG. 云平台会在改变结构时调用该接口. 更新算法如算法 1 所示，在删除节点或边时只标记他的删除时间，具体的删除操作在查询时完成.

算法 1 更新算法

```
Algorithm Update
Input: operation, v2eMap
Output: v2eMap
1 switch operation
2 case addNode
3      add v and relative e
3 case deleteNode
4      v.deletetime ← currentTime
5      for each e in v2eMap.get(v)
6            e.deletetime ← currentTime
7 case migrateNode
8      deleteE.deletetime ← currentTime
9      add e to v2eMap.get(v)
```

② 查询(research): 返回某节点的相关节点并将图中权值大于两倍时间窗口的对象删除. 输入为某节点标识，输出为相关节点的列表. 实际上是在 DAG 中找出以输入节点为根的子树中的所有节点.

算法 2　相关节点查询算法

```
Algorithm RelativeNodeSearch
Input: selectedNode, n2vMap, v2eMap
Output: relativeNodeList
1 Clean()
2 selectedV ← n2vMap.get(selectedNode)
3 relativeVList ← subgraphSearch(selectedV)
4 add relativeVList to relativeNodeList
5 return relativeNodeList
Algorithm subgraphSearch
Input: selectedV
Output: relativeVList
1 eList ← v2eMap.get(selectedV)
2 for each e in eList
3      latterV ← vk in e
4      add subgraphSearch(latterV) to relativeVList
5 end for
6 return relativeVList
Algorithm Clean
1 for each v in V
2      if currentTime - v.deleteTime>=2w
3            delete v
4      end if
5 for each e in E
6      if currentTime - e.deleteTime>=2w
7            delete e
8      end if
```

4.2.2 日志合并模块

该模块负责将出错关键节点相关的所有节点在故障时间窗口内的日志合并起来，按时间排序.

算法 3 日志检索算法

```
Algorithm BinarySearch
Input: logArray, num,  startTime, endTime
Output: resultLogList
1 start←0 end←num-1
2 while start<end  do
3      mid←start+(end-start)/2
4      if logArray[mid].time>=startTime
5      then
6            if logArray[mid-1].time<startTime
7                  then break
8            end if
9            end←mid
10     else
11           if logArray[mid+1].time>=startTime
12                 then mid←mid+1  break
13           start←mid
14     end if
15 end while
16 while logArray[mid].time<=endTime do
17     add logArray[mid] to resultLogList
18     mid←mid+1
19 end while
20 return resultLogList
```

计算节点中的出错日志会给出关键节点的标识，调用节点依赖关系维护模块的查询接口，找出相关的节点标识，然后分别在这些节点的日志中查询时间窗口内的日志.

日志天然按时间排序，获取日志时，以二分查找算法完成日志的快速检索，查找操作时间复杂度由 O(n)降为 O(logn). 快速检索算法即算法 3 所示.

4.2.3 故障分类模块

故障分类模块本质上是一个决策树分类器，本文选用 C4.5 决策树的著名实现, Weka 的 J48 决策树作为我们提取故障特征的工具. C4.5 算法是 ID3 算法的改进，现在已经成为最经典的决策树构造算法. Weka 的全名是怀卡托智能分析环境，是一款基于 Java 环境的开源的机器学习及数据挖掘工具. 经过十多年年的发展历程，Weka 是现今最完备的数据挖掘工具之一，而且被公认为是数据挖掘开源项目中最著名的一个，近年来大量与数据挖掘相关的研究工作均选用 Weka 作为基本平台. 在 Weka 中使用 C4.5 的相关配置参数如表 1 所示.

表 1　C4.5 参数描述表

参数名	参数描述	默认值
binarySplits	二进制分裂	False
confidenceFactor	置信因数	0.25
minNumObj	每个叶子的最小实例数	2
reducedErrorPrunning	是否使用减少-误差方式剪枝	False
numFolds	减少-误差修剪时修剪的数据量	3
seed	减少-误差修剪时随机数据的种子	1
unpruned	是否需要修剪	False

由于使用 C4.5 剪枝算法，所以 numFolds, reducedErrorPruning 和 seed 皆不需进行设置. 原始数据属性已经为二维，因此 binarySplits 也不需设置. 鉴于 C4.5 算法会在运行过程中自动剪枝，所以 unpruned 的设置无用. 因此，比较重要的参数为 confidenceFactor 和 minNumObj. 由于我们最终的研究目标为通过过滤提升故障定位的有效性和性能，所以不讨论不同参数对特征模型精确率的影响. 在实验过程中均使用 Weka 提供的默认值，实际结果在第五章可见，由结果验证其有效性.

5　实验验证

本文通过在 OnceCloud 虚拟化云平台上的实验验证了故障定位方法和系统的有效性和性能. 在系统运行时人为注入故障，根据日志建立事件模型，随后再次注入故障，根据之后收集的系统日志信息进行故障识别和定位.

5.1 测试用例

测试包含 30 轮，每轮 6 个测试用例，每个测试用例分别模拟了不同组件不同类型的崩溃场景. 具体测试用例如表 2 所示.

表 2 测试用例表

测试用例名称	故障发生组件	类型
STfile	存储	文件损坏
STservice	存储	服务中止
NetFail	网络	网络故障
Memory	计算	内存紧张
ServantFail	计算	从节点宕机
MasterFail	计算	主节点宕机

测试用例涵盖了故障发生可能的三种组件，其中存储组件的两类故障和网络组件的故障在计算节点的出错日志应该是相同的，在进行定位时需要采集相应组件的信息. 而计算组件的故障可以通过计算节点的日志直接进行定位.

5.2 测试结果

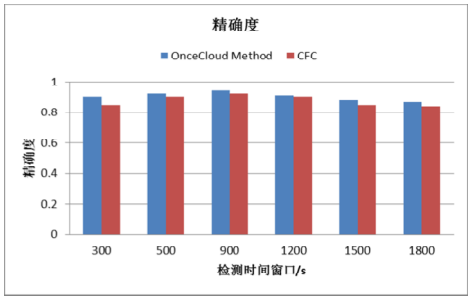
5.2.1 有效性测试

为了评估故障定位方法的有效性，本文将实验结果与 Zheng Z[5]所提出的处理框架 CFC 进行了对比，主要对比了 CFC 以及本文方法的精确度(precision，即正确检测结果与所有检测出结果的比率)和召回率(recall，即正确检测结果与所有注入故障数量的比例). 同 CFC 方法一样，时间窗口分别设定为 300s，500s，900s，1200s，1500s，1800s.

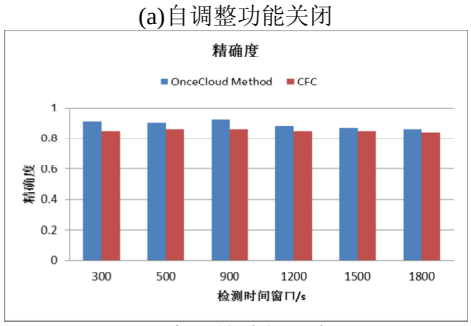
为了能够检验空间拓扑变化对于故障定位方法的影响，本文对比了云平台自调整、高可用功能开闭情况下的精确度和召回率. 当云平台开启自调整和高可用功能时，云平台上的应用需求变化和某些故障都会引起虚拟化环境空间拓扑的变化.

图 4 和图 5 显示，当自调整功能关闭，即虚拟化环境空间拓扑不发生变化的时候，不管是在精确度还是召回率上本文所采用的方法都和 CFC 方法不相上下；而当自调整和高可用功能开启，虚拟化环境会随着应用负载的加重而调整资源分配，高可用功能会在物理节点发生宕机时自动迁移虚拟机时，本文的方法显示出了更好的适应性.

当虚拟化环境空间拓扑随时间变化时，CFC 方法在依照时间维度进行日志过滤时，会丢掉由不同宿主物理机的日志共同构成某故障的特征类型，影响是不能检测出故障的发生. 因此尽管精确度上差别不大，在召回率上明显低于本文的方法.



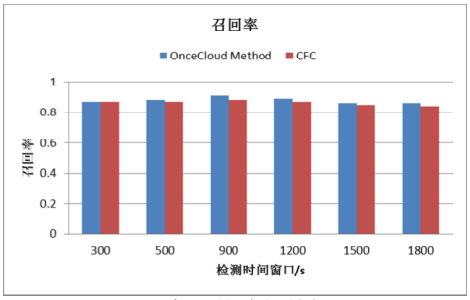
(a)自调整功能关闭



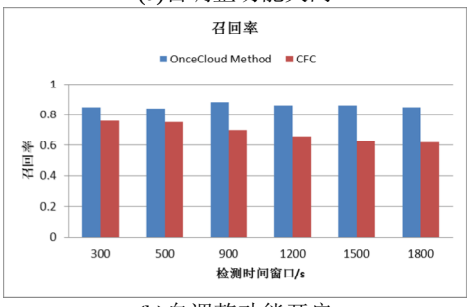
(b)自调整功能开启

图 4 精确度对比图

从图中亦可以看出随着检测时间窗口的加大，方法之间的差距越发明显，这是因为检测时间窗口的加大也加大了在检测时间窗口内虚拟化环境空间拓扑发生变化的可能性，符合预期.



(a)自调整功能关闭



(b)自调整功能开启

图 5 召回率对比图

以上实验充分保证了在虚拟化环境空间拓扑随时

间变化时，本文的方法保证了故障定位的有效性.

5.2.2 性能对比测试

通过计算平均一次测试故障定位所需要的时间，在不同系统的规模情况下进行对比. 性能对比测试在云平台提供的自调整、高可用等功能打开状态下进行.

故障定位需要的时间的定义为故障检验阶段从故障注入到系统返回结果所需要的时间，系统规模由节点个数定义.

图 6 为固定物理节点数为 10 的情况下，改变虚拟机个数，系列 Filter-On 为进行了过滤的结果，Filter-Off 为未过滤的结果. 可以看到随着虚拟节点个数的增加，定位时间并未有太大的变化，原因是目前只在物理节点上收集日志. 而可以看出经过过滤的结果反而更差一些，这是因为由于要在内存中维护节点依赖关系图，并在定位过程中还需要进行检索的原因所致.

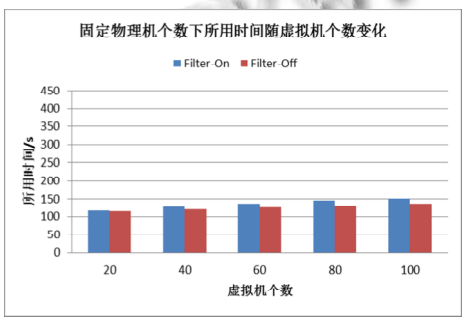


图 6 耗时随虚拟机个数变化图

图 7 为固定虚拟节点数为 100 的情况下，改变物理节点个数，系列 a 为进行了过滤的结果，b 为未过滤的结果. 可以看到随着物理节点个数的增加，未过滤

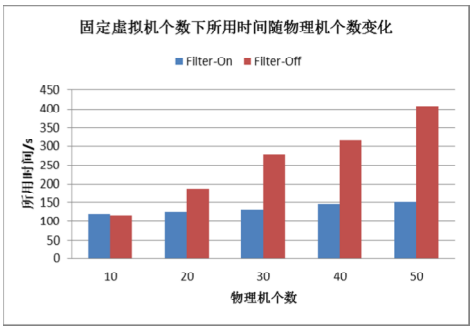


图 7 耗时随物理机个数变化图

的故障定位时间是近乎线性增长，而过滤后的结果，虽然也有增加，但是能够看出，在系统规模增大的情况下，本方法能够保证故障定位时间的稳定.

6 总结

在云平台的运行维护中，虚拟化环境的故障定位对于容错极其重要. 本文提出了一种基于节点间依赖关系的日志过滤方法，从而为之后基于决策树的故障定位方法提供了良好的支持. 通过在 OnceCloud 系统上对于精确度和召回率的检验，以及对比是否使用过滤方法下故障定位时间随系统规模的变化，本文提出的方法能够在较大系统规模的多变的虚拟化环境下实现有效和高效的故障诊断.

参考文献

1 Mellp G. The NIST definition of cloud computing. National Institute of Standards and Technol, 2011

2 饶翔,王怀民,蔡华,周琦,孙廷韬,史殿习,尹刚.云计算系统中基于噪声模板跳表的日志过滤方法.通信学报,2011,32(7): 103–113.

3 饶翔,王怀民,陈振邦,周扬帆,蔡华,周琦,孙廷韬.云计算系统中基于伴随状态追踪的故障检测机制.计算机学报,2012,35(5):856–869.

4 Yuan D, Mai H, Xiong W, Tan L, Zhou Y. SherLog: Error diagnosis by connecting clues from run-time logs. ACM SIGARCH Computer Architecture News, 2010, 38(1): 143–154.

5 Zheng Z, Lan Z, Park BH. System log pre-processing to improve failure prediction. IEEE/IFIP International Conference. 2009. 572–577.

6 Liang YL, et al. Filtering failure logs for a bluegene/l prototype. Dependable Systems and Networks. 2005.

7 Nguyen H, Shen Z, Tan Y, Gu X. FChain: Toward black-box online fault localization for cloud systems. Distributed Computing Systems (ICDCS). 2013 IEEE 33rd International Conference. 2013. 21–30.

8 Yuan D, Park S, Huang P, Liu Y, Lee MM, Tang X. Be conservative: enhancing failure diagnosis with proactive logging. OSDI. 2012. 293–306.

9 Barham P, Dragovic B, Fraser K, Hand S, Harris T. Xen and the art of virtualization. ACM SIGOPS Operating Systems Review, 2003, 37(5): 164–177.

10 Kivity A, Kamay Y, Laor D, Lublin U, Liguori A. Kvm: the Linux virtual machine monitor. Proc. of the Linux Symposium, 2007, 1: 225–230.