基于宣告式网络的网络溯源系统的设计与实现

高 翔[1] 王 晓[1] 王 敏[2]
(西北工业大学计算机学院 西安 710072)[1] (空军工程大学电讯工程学院 西安 710077)[2]

摘 要 网络的举证分析、错误诊断在网络管理和安全方面正发挥着越来越重要的作用。这就要求网络管理系统具有网络溯源的功能。网络溯源可以用于跟踪信息在网络上流传的轨迹,确定信息数据来源。提出了一个网络溯源系统(NPS)框架的设计与实现,该框架可以支持在大规模的分布式环境中获得网络溯源,采用了新近提出的宣告式网络技术来有效地维护和查询分布式网络溯源。该框架采用基于引用的方式来传递溯源信息,采用有向无环图来表示溯源信息,在分布式网络中实现了高效的网络溯源。在ns-3构建的模拟网络中进行了仿真实验,实验结果表明该网络溯源系统框架可以有效地支持一个大规模分布式网络的溯源计算,与传统方法相比显著地减少了带宽的开销。

关键词 网络溯源,宣告式网络技术,分布式查询处理,查询优化
中图法分类号 TP393 **文献标识码** A

Design and Implementation of Network Provenance System Based on Declarative Networking

GAO Xiang[1] WANG Xiao[1] WANG Min[2]
(School of Computer Science, Northwest Polytechnical University, Xi'an 710072, China)[1]
(School of Communication, Air Force Engineering University, Xi'an 710077, China)[2]

Abstract Network forensic analysis and fault diagnosis are becoming increasingly important in network management and network security domain. This requires network management system has the ability to query network metadata. For instance, network provenance can be used in tracking the path of dataflow through the network to obtain the source of data. This paper presented the design and implementation of a network provenance system (NPS) framework. The framework is used to support the full range of functionality required for enabling forensics in distributed systems. We adopted the declarative networking technique in the networking domain to maintain and query distributed network provenance. The framework adopts a reference-based approach to transfer provenance information and a cyclic graph to represent the provenance information, implementing efficient network provenance in distributed network. Simulation experiments were conducted in simulated network. The experiment results indicate that our network provenance system can support provenance process in a large-scale distributed network and significantly reduce bandwidth cost compared to traditional approach.

Keywords Network provenance, Declarative networking, Distributed query processing, Query optimization

1 网络溯源概述

网络溯源是一种查询网络元数据的技术。网络溯源描述了执行分布式协议时网络状态的历史和导出,被广泛应用于网络管理、诊断和安全性分析。典型的网络溯源应用包括识别特定信息的来源,推导信息在网络中的遍历路径的过程。文献[1]提出了一种根据语法计算溯源信息的方案并将结果以通用关系型数据库数据模型或层次性数据如XML的形式组织起来。独特之处在于将溯源原因(源数据对现有数据的影响)和溯源位置(数据是从数据库的哪一源数据处取出)区分开来。文献[2]提出VisTrails系统的原型,该系统为仿真模拟与数据挖掘提供可视化溯源支持。它允许科研工作者高效地通过可视化结果研究科学数据。文献[3]中提出了在分布式元组更新流中增量递归维护的方法:新的流数据被视作插入操作,过期元组被视为删除操作。提出了一系列维护元组导出和数据溯源的方法,即通过聚合选择来对无关元组进行剪枝以减少通信量;并将这一思想在模拟无线网络中进行了验证。Wenchao Zhou[4]提出了Exspan(可扩展的溯源感知的网络系统)设计与应用,Exspan是能在分布式环境中有效进行网络溯源的通用可扩展框架平台,并为网络溯源存储定义了一种分布式模型,其用数据溯源的理念来解释网络中存在的各种状态。Grigoris Karvounarakis[5]提出了一种基于元组、半环溯源的ProQL语言,它能够解决溯源存储、维护和查询等相关问题。

到稿日期:2013-06-01 返修日期:2013-09-19 本文受国家科技支撑计划(2012BAB15B01)资助。
高 翔(1974—),男,博士,副教授,主要研究方向为高可信网络,E-mail:gaoxg@nwpu.edu.cn;王 晓(1989—),男,硕士生,主要研究方向为高可信网络,E-mail:wangxiao5018@163.com(通信作者);王 敏(1974—),女,博士,副教授,主要研究方向为信息处理。

本文第1节介绍了网络溯源的研究现状；第2节介绍了宣告式语言的特点；第3节介绍了基于宣告式语言的网络溯源框架的基本结构；第4节提出了分布式溯源的数据模型和维护模块；第5节给出了分布式溯源查询的机理及查询定制方法；第6节对框架进行了仿真实验并对结果进行了分析；最后总结了我们的工作并对网络溯源技术的未来发展进行了展望。

2 宣告式语言

网络的大规模分布式特性给网络协议的设计开发带来诸多挑战，需要在保证协议的可扩展性与灵活性的同时兼顾健壮性与效率。网络协议设计者面临的最基本问题是如何脱离繁琐的协议实现细节而将主要精力放在协议的功能与概念设计上。宣告式编程语言是一种使我们能在高层具体指定“做什么”而不是“如何做”的语言，宣告式编程着力于解决传统分布式编程与并行编程的问题，可使编码过程简化并显著减少代码量。

宣告式语言代表性的有 Netlog，Overlog，NDlog 等。NDlog 是在推导式数据库查询语言 Datalog 的基础之上提出的描述路由协议的宣告式网络程序设计语言。OverLog 是在 NDlog 语言的基础之上增加了对网络中数据软状态（生存周期）的描述。在无线网络环境中，由于网络节点移动以及网络节点的失效，网络具有很强的动态性。为了给无线网络协议设计者提供好的编程抽象，形式化定义了 Netlog 语言。在本文中我们选用 NDlog 语言。宣告式编程语言可以紧凑简洁地表述许多典型路由协议（距离向量，向量路径，动态源路由，链路状态，多播），通常情况下代码只有很少几行。

网络协议以 NDlog 语言表述并分布在各个节点，每个节点将 NDlog 规则编译为分布式的数据流。当协议执行时，这些数据流在节点之间交换信息和网络状态，最终达到稳定，如图1所示。

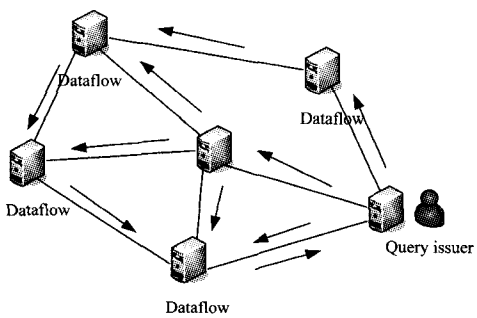


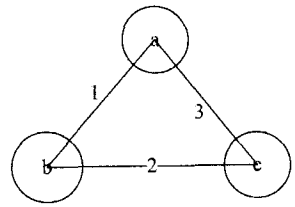
图1 分布式查询数据流

Mincost 协议在图2的示例网络中执行，每个节点在执行前已拥有各自的本地 Link 表。利用 NDlog 语言表述的程序由一系列规则组成，规则具有形式：$p:-q_1,q_2,\cdots,q_n$。p 称为规则首部，q_i 构成规则主体。q_i 之间的逗号表示逻辑与操作，q_i 是谓词或者是函数，每个谓词（关系）有一个主键，其中包含一组唯一标示关系中每个元组的字段。只有在所有的函数和谓词都满足的情况下，这条规则才被执行并推导出首部。以下是用 NDlog 语言表示的最小路径代价路由算法。

sp1 pathCost(@S,D,C):- link(@S,D,C).

sp2 pathCost(@S,D,C1+C2):-
link(@S,Z,C1),minCost(@Z,D,C2).

sp3 minCost(@S,D,min<C>):-pathCost(@S,D,C).



link(@source,destination,cost)

source	destination	cost
a	b	1
a	c	3
b	a	1
b	c	2
c	a	3
c	b	2

图2 示例网络拓扑

规则 sp1 直接从本地 link 表中生成 pathCost 元组，规则 sp3 根据本地存储的 pathCost 表选出最小代价元组 bestPathCost，规则 sp2 利用上一轮邻居节点生成的 bestPathCost 元组与本地 link 元组生成新的 bestPathCost 元组。在每个谓词的地址字段前有一个@标示符，标明元组的实际存储地址，地址标示符使 NDlog 语言，具有表示分布式计算的能力。

3 网络溯源框架

本文提出一个基于宣告式技术的用于在分布式环境中进行高效溯源的通用框架，通过该框架可以解释任何一项网络状态变化的原因，也可用于网络中违法行为的网络溯源和调查取证。

3.1 溯源信息粒度

溯源框架通过3个粒度级别来调节溯源信息详尽程度及其对网络性能的影响。元组级别溯源包括导出过程中的所有中间元组，元组级别溯源在网络网协议调试中具有重要作用，所包含的溯源信息最详尽，但引起的通讯开销最大；节点级别溯源通过溯源信息来查询哪些节点参与了元组的导出；信任域级别中，一个信任域中的若干节点共享一个域标识符，溯源信息仅记录参与元组导出的信任域。

3.2 溯源信息表示

可以将溯源信息表示为溯源图（Provenance Graph）的形式。如图3所示，溯源图反映了导出元组与基元组的关系，溯源图中每个结点表示一个带地址标示符的数据关系操作，图中每一条边表示中间计算结果。

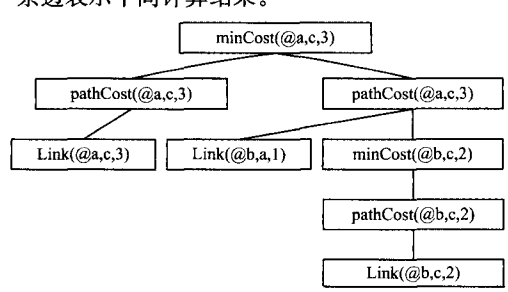


图3 示例溯源图

3.3 溯源信息分布

溯源信息可以被集中或分布式地存储在网络中。在集中式溯源中，一个元组携带自身的所有溯源信息。为维护完整的集中式溯源，所有的溯源信息必须通过一台中央服务器进行转发。集中式溯源可利用传统集中式数据库来实现，通过修改关系操作符使协议在执行的同时完成溯源信息的生成。集中式方案在中央服务器处形成单一瓶颈和高度聚合的带宽使用，不适合部署于地理位置高度分散和拥有大量节点的大规模分布式系统。

另一种方式是分布式地存储溯源信息，在基于值的分布式溯源中，在节点之间交换信息时每一个导出元组必须携带自身的全部溯源信息，这种方法的好处是查询反应快，但是存储开销和通讯开销相对较大。而基于引用的分布式溯源在协议执行时生成可供反向追踪的引用标签，溯源信息平均分布在多个网络节点上，元组在交换时只携带若干基本引用标签。在查询阶段，利用分布式查询协议递归地遍历每个节点上的溯源信息表来增量地构建溯源图。

4 溯源信息维护

4.1 数据模型

溯源系统使用有向无环图 $G(V,E)$[7,8] 作为溯源信息的数据模型。V 为顶点集，顶点集包含基元组和计算结果。E 为边集，边集表示元组与计算结果之间的一次规则执行，从元组到计算结果的边表示元组为规则的输入，从计算结果到元组的边表示元组为规则的输出。顶点集使用 VID 与 RID 来唯一地标示。为了唯一地标示溯源图中的每一个元组顶点，为每一个元组顶点生成一个经过哈希加密的顶点 ID。例如元组顶点 link(@X,Y,C) 的 VID 号被计算为 VID=SHA1("link"+X+Y+C)。对于规则执行顶点，一个规则 ID 具有形式 RID=SHA1("r2"+X+t1+t2)，表示规则 r2 利用输入元组 t1 和 t2 在节点 X 被执行。

4.1.1 存储模型

使用 Prov 表和 ruleExec 表维护溯源信息和查询请求。Prov 表储存溯源信息，每一表项表示一个元组推导，具有形式 prov(@Loc,VID,RID,RLoc)，表示位于 Loc 节点的元组 VID 是由位于 RLoc 节点的规则执行顶点 RID 生成的，Prov 表通过位置标示符 Loc 分布式地存储在多个节点上。ruleExec 表存储规则执行的元数据，字段 RLoc 指明规则的驻留位置，每一条表项表示一次规则执行，具有形式 ruleExec(@RLoc,RID,R,VIDList)。表 1 是一个图 2 中网络拓扑对应的 Prov 表，在实际网络中，表 1 中的表项是按 Loc 字段分布式地存储在各个节点之上的。例如第一项和第二项，这两个表项表示元组 pathCost (@a,c,3)有两个方向的导出，一个方向由规则执行顶点 RID2 得出，另外一个由规则执行顶点 RID3 推导出。在得到 RID 和 RLoc 字段后，先由 RLoc 确定规则执行节点，再在对应的表 2 中根据 RID 字段查找 VIDList 集合，最终递归查询停止在基元组。溯源信息以遍历顺序相反的方向返回到查询节点。

表 1 示例 Prov 关系表

Loc	VID	RID	RLoc	Derivation
a	VID5	RID2	a	pathCost(@a,c,3)
a	VID5	RID3	b	pathCost(@a,c,3)
a	VID7	RID5	a	minCost(@a,c,3)
b	VID4	RID1	b	pathCost(@b,c,2)
b	VID6	RID4	b	minCost(@b,c,2)

表 2 示例 ruleExec 关系表

RLoc	RID	R	VIDList	Derivation
a	RID2	sp1	(VID3)	pathCost(@a,c,3)
a	RID5	sp3	(VID5)	minCost(@a,c,3)
b	RID1	sp1	(VID1)	pathCost(@b,c,2)
b	RID3	sp2	(VID2,VID6)	pathCost(@a,c,3)
b	RID4	sp3	(VID4)	minCost(@b,c,2)

4.1.2 溯源信息传递方式

在基于值的分布式溯源中，每个待发送的元组携带自身的全部溯源树(即相关的 Prov 表项和 ruleExec 表项)。基于值的溯源方案导致较高的通讯开销，然而其适用于在特定的网络管理应用中，节点根据元组的内容决定接受或拒绝一个信息元组。在基于引用的分布式溯源中，每个元组需要携带 20 字节的 RLoc 和 RID 字段以通过分布式查询来构建溯源信息。在溯源查询阶段，溯源图以分布式递归的方式被增量地遍历。给定一个 VID 值，通过查询 Prov 表中的 RLoc 字段得到对应的 VIDList 字段，随后根据内容在 ruleExec 表中查找对应的 RID 字段，最终查询停止在基元组，溯源信息被增量地返回以构建溯源树。

4.2 分布式溯源维护

在增量的视图维护中，每次规则执行和新元组导出都会生成新的 Prov 表项和 ruleExec 表项。同理，每次基元组的删除都会导致对应的 Prov 表项和 ruleExec 表项的连锁删除。

溯源框架使用 5 条 NDlog 规则在协议执行的同时生成分布式存储的溯源信息。例如，MinCost 协议中规则 sp2 被自动重写为 5 条维护规则 R1－R5。下面给出了部分维护规则(规则 R4 和规则 R5)。

```
R4 ruleExec(@RLoc,RID,R,List) :-
  ePathCostTemp(@RLoc,S,D,C,RID,R,List)
R5 prov(@S,VID,RID,RLoc) :- ePathCost(@S,D,C,RID,RLoc)
  VID=f_sha1("pathCost"+S+D+C)
```

前 3 个溯源信息维护规则(R1－R3)生成 MinCost 协议所需要的 S,D 和 C 字段，同时这 3 个规则生成 4 个溯源维护字段：RLoc,RID,R 和 LIST。这 4 个字段组成了 R4 规则中的 ruleExec 表项。随后规则 R4 生成的 ePathCost 事件元组依据地址标示符被送往节点 S。在节点 S，到达的 ePathCost 事件元组被用于执行协议和生成 Prov 表项(规则 R5)。

5 溯源信息查询

5.1 分布式递归查询

以溯源图中元组 bestPathCost 的溯源信息查询过程为例。对照表 1，该元组对应的 VID 值为 VID7，首先初始一个 VID7 查询，按 Prov 表第 3 项找到其对应 RID 值为 RID5。随后依据 RLoc 和 RID 在表 2 中找到第 2 项，表项中对应的

VIDList 为 VID5，返回到 Prov 表中找到两条符合条件的表项(VID5 拥有两个 children)，同时触发了两个 VID5 查询(VID5 由 ruleExec 表项的 VIDList 得到)，其过程类似于 VID7 查询。而 VID5 查询会触发新的 VID 查询直到该 VID 为基本元组为止。直至 VID5 的所有 children 都返回了 Prov 溯源信息后，RLoc 节点将整合后的溯源信息发回给 Request 节点。

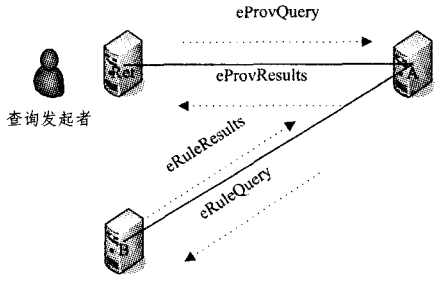


图 4　分布式查询处理

以上是溯源信息查询的大致过程，具体实施时，通过 2 条基本规则(edb1，c0)和 8 条递归规则(idb1－idb4，rv1－rv4)来实现查询功能。规则 edb1 用于返回基本元组溯源信息，规则 c0 用于统计一个 VID 对应的 children 数目。在执行查询时共有两对查询/回复事件元组(eProvQuery/eProvResults 和 eRuleQuery/eRuleResults)，分别对应对 Prov 表和 ruleExec 表的操作。

```
idb1 pResultTmp(@X,QID,Ret,VID,f_empty()) :-
     eProvQuery(@X,QID,VID,Ret),
     prov(@X,VID,RID,RLoc),RID!=NULL
idb2 eRuleQuery(@RLoc,RQID,RID,X) :-
     eProvQuery(@X,QID,VID,Ret),
     prov(@X,VID,RID,RLoc),
     RQID=f_sha1(QID+RID)
idb3 pResultTmp(@X,QID,Ret,VID,Buf) :-
     eRuleResults(@X,RQID,RID,Prov),
     pResultTmp(@X,QID,Ret,VID,Buf1),
     RQID=f_sha1(QID+RID),
     Buf=f_concat(Buf1,Prov)
idb4 eProvResults(@Ret,QID,VID,Prov) :-
     pResultTmp(@X,QID,Ret,VID,Buf),
     numChild(@X,VID,C),C=f_size(Buf),
     Prov=f_pIDB(Buf,VID,X)
```

idb 规则(idb1－idb4)用于对 Prov 表的操作，Request 节点提出一个查询请求 eProvQuery(@X,QID,VID,Ret)，其中 QID 为与其他查询相区分的查询号。若查询的 VID 为基本元组，则执行 edb1 查询即结束，否则，规则 idb1 根据元组 eProvQuery 提供的地址在对应节点初始化溯源信息缓存。规则 idb2 向 RLoc 节点(协议执行节点)发送事件元组 eRuleQuery，根据 RID 字段在 ruleExec 表中检索符合条件的表项。rv 规则(rv1－rv4)用于 ruleExec 表中规则执行节点的查询，执行过程类似于 idb 规则。

idb 规则和 rv 规则被轮番触发直到所有子节点被遍历。随着查询的进行，eProvQuery 事件元组被递归地由查询发起节点传播到基元组以构建整个溯源图。最终，溯源查询模块交替访问 Prov 表和 ruleExec 表直到获得溯源图。

5.2　查询定制

5.1 节中的溯源查询规则调用了 3 个用户定制函数：f_pEDB函数，f_pIDB 函数，f_pRule 函数。在框架原型中，f_pEDB函数被规定返回基元组本身；f_pIDB 函数返回一个元组的所有可能的中间导出元组；f_pRule 函数用于规则执行实例，返回输入元组的乘积。可以对原型框架中的用户定制函数进行修改，使其返回需要的结果。基于 5.1 节的查询框架，在这一节讨论如何对用户定制函数进行修改以满足不同的查询需求，下面讨论 3 种查询定制。

可以规定用户定制函数返回一个特定元组的所有导出的数目。将 3 个用户自定义函数定义如下：f_pEDB 等价于整数 1，表示每个 EDB 元组有一个导出。对于一个中间元组，f_IDB 的导出数目可被计算为所有中间结果的和。对于规则执行实例，f_pRULE 被定义为中间结果的乘积。

图形投影定制允许用户在查询溯源信息时只返回部分受信任节点的溯源信息。例如在多管理域中，一个节点只生成或接收自己所在的信任域中的溯源信息，溯源图投影可以通过在查询模块中添加额外的限定条件来实现，例如当规则 idb2 被触发时，只将 eRuleQuery 元组发给特定的分支而并不转发给所有分支。

查询定制的最后一个例子是将溯源信息元组中的缓存字段替换为一个经过哈希后的顶点值，与原型不同，节点间传播的不是溯源表项而是一个哈希序列值，如果用户需要进一步获得顶点值的详细信息，可以通过查找本地 Prov 表来获得对应的表项。

6　仿真与评估

我们使用 NS-3 网络模拟器[12]来进行模拟实验。在实验与验证阶段，在 Ubuntu10.14 下，我们使用 3.5.1 版本的 NS-3 模拟器来构建网络拓扑结构，使用 Rapidnet 编译器[14]来编译 NDlog 程序。Rapidnet 是一个基于 NS-3 网络模拟器的用于网络协议仿真和实现的开发包。Rapidnet 在 NS-3 的基础上整合了声明式组网引擎以执行宣告式程序，在 Rapidnet 中协议以 NDlog 代码的形式表述，随后 Rapid 编译器将 NDlog 代码自动转换为在 rapidnet 函数库上运行的 C++代码。

6.1　原型分析

我们比较了溯源框架原型在两个示例协议上的运行情况。MinCost 协议用于得出各个节点之间若干最短路径 minCost(@S,D,C)。PathVector 协议用于得出各个节点之间的最佳路径 bestPathCost(@S,D,C,P)，其中 P 字段是一个路径矢量，存储着网络中的节点链。实验在不同的节点规模下进行，分别记录了节点间的平均通信量与平均包大小。在实验前提前为每个节点设置好各自的本地 link 信息表，实验的终止条件是不再有新的路径元组被导出或不再有新的事件元组到达。

图 5 和图 6 显示了不同网络规模下分别执行 MinCost 协

议与 PathVector 协议所引起的平均通信开销。图 7 和图 8 显示了不同网络规模下分别执行 MinCost 协议与 PathVector 协议的平均数据包大小。其中原协议(图中 MinCost 和 PathVector)的通信开销均小于部署溯源功能的协议(图中 MinCost-Prov 和 PathVector-Prov),原协议的平均数据包大小均小于携带溯源功能的协议,这是由于网络溯源会引起额外的通讯开销,溯源框架需要在执行原协议的同时生成供用户查询所需的溯源维护信息。图 9 和图 10 显示了在节点频繁进出与连接失效的情况下分别执行 MinCost 协议与 PathVector 协议所引起的平均通信开销,溯源框架对网络抖动情况具有适应性,并没有引起显著的通信开销,例如图 9 中 MinCost 与 MinCost-Prov 近似重合。

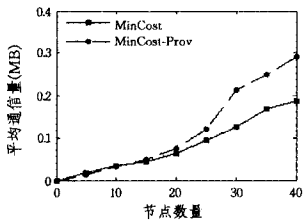


图 5 MinCost 协议的平均通信开销(MB)

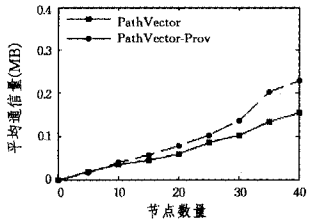


图 6 PathVector 协议的平均通信开销(MB)

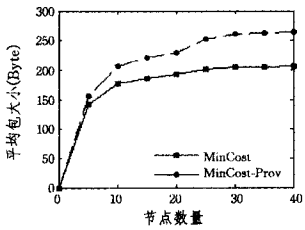


图 7 MinCost 协议平均数据包大小(Byte)

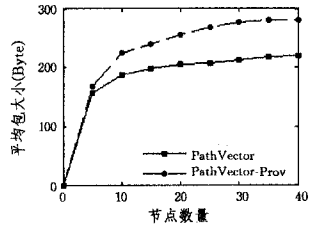


图 8 PathVector 协议平均数据包大小(Byte)

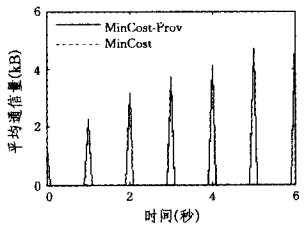


图 9 网络抖动下 MinCost 协议平均通信量(kB)

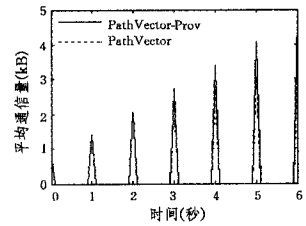


图 10 网络抖动下 PathVector 协议平均通信量(kB)

6.2 查询优化与定制

在使用宽度优先顺序(BFS)时,必须遍历一个顶点的所有导出,一个 VID 顶点的多个分支被同时查询。在使用深度优先顺序(DFS)时,顶点的一个分支被遍历到叶子节点,随后返回继续下一个分支的遍历。基于这一遍历特点,DFS 适用于检测一个元组的导出元组数量或在一次导出中参与节点的数量等基于阈值的查询。DFS 与 BFS 相比,增加了查询延迟,减少了通信开销。

遍历顺序实验在不同节点规模下进行,我们比较了两种遍历顺序下的节点间平均通信开销与查询完成时间。查询定制实验通过增加用户定制函数来完成特定的查询要求,我们试验了 5.2 节中最后一个例子以观察查询定制实验的各项参数。查询优化实验与查询定制实验均使用 MinCost 协议作为基本实验协议。

我们在不同节点数量下对两种遍历方式进行了实验。图 11 显示了 30 个节点的网络上不同遍历顺序对平均通信开销的影响;图 12 显示了两种遍历方式下溯源信息的查询平均时间。实验表明基于深度优先顺序的遍历方式的通信开销与基于广度优先顺序的遍历方式的通信开销大致相等,但收敛速度较后者快。

图 13 显示了查询定制实验在 30 个节点规模下的平均通信开销(图中实线 Prototype 为溯源原型,虚线 Modified 为重写用户定制函数后的溯源框架);图 14 显示了不同节点下查询定制方案的平均通信开销。模拟结果表明,由于对返回元组中的溯源信息字段进行了压缩,与溯源原型相比,查询定制方案进一步减少了节点间的平均通信开销。

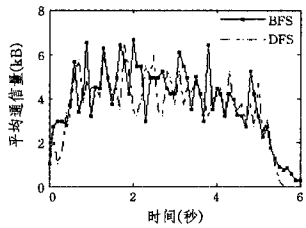


图 11 不同遍历顺序下 MinCost 协议平均通信量(kB)

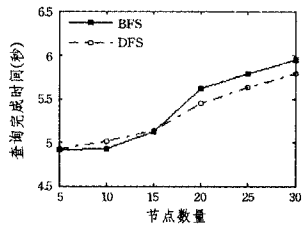


图 12 不同遍历顺序下 MinCost 协议的查询完成时间

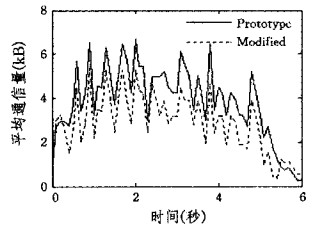


图 13 MinCost 协议查询定制的平均通信量(kB)

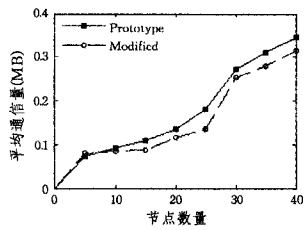


图 14 不同网络规模下查询定制策略对 MinCost 协议平均通信量(kB)的影响

结束语 本文提出一个用于在分布式环境中进行高效溯源的通用框架,通过该框架可以解释任何一项网络状态变化的原因,可对网络中的违法行为进行网络溯源和调查取证。溯源框架使用两个关系数据表来存储溯源信息,使用 5 条溯源信息维护规则来生成与更新溯源信息,并使用 8 条溯源查询规则来接受用户查询。

在对宣告式溯源技术的验证中,我们使用 NS3 网络模拟器构建了实验环境,进行了多个实验。溯源实验结果表明,与同类方法相比,宣告式溯源技术在保证高效溯源的同时有效地降低了通信开销。查询优化实验表明,在基于阈值的条件查询中深度优先遍历顺序更有效率,调节溯源信息粒度可以有效地降低查询返回时间和通信开销。在进一步的研究中,我们将探索在不可信环境中搭建高效溯源框架的方法,考虑溯源框架与现有错误探测技术[15,16]的结合,以确保溯源信息的机密性和可靠性。

(下转第 194 页)

在图9中，Baidu PCS的数据准备时间 TC_{pre} 随数据大小的变化不甚明显，在18～22ms周围波动，而Kuaipan的 TC_{pre} 随着数据大小的增加有明显的增加趋势，并且从图10可以看出，Kuaipan的数据准备时间 TC_{pre} 波动性远远大于Baidu PCS，Baidu PCS的 TC_{pre} 围绕其均值做小范围的变动。Baidu PCS TC_{pre} 随数据大小变化不明显的原因是Baidu PCS在读取若干数据块之后就发送大网络，而Kuaipan则是在读完整个数据之后才发送到网络上，所以Baidu PCS表现出 TC_{pre} 变化不明显，而Kuaipan的 TC_{pre} 表现出随数据大小增大而增大的情况。

综上，在读数据的过程中，Baidu PCS不论是在网络传输 TC_{trans} 还是在数据准备 TC_{pre} 上均优于Kuaipan，并且其稳定性也优于Kuaipan。在排除网络引起的干扰之后，Baidu PCS在读取用户数据时的性能要优于Kuaipan。

结束语 本文通过分析云存储客户端与云存储服务端通信IP报文，找出读写数据的几个关键时间点，计算出读写数据各个阶段所用时间，分析这些时间开销与数据大小的关系，比较了Baidu PCS和金山Kuaipan之间对应时间开销的大小，得出在不考虑网络带宽的情况下，Baidu PCS的性能优于金山Kuaipan。从实验结果可以看出，本文提出的读写时间开销模型能够很好地评测不同的云存储系统。可以根据侧重点不同，使用本文所提模型观测影响云存储系统的各个因素，对云存储进行完整的评测。

参 考 文 献

[1] Agarwal D, Prasad S K. Azurebench. Benchmarking the storage services of the Azure Cloud Platform[C]//Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International. IEEE, 2012: 1048-1057

[2] Benefico S, Gjeci E, Gomarasca R G, et al. Evaluation of the CAP Properties on Amazon SimpleDB and Windows Azure Table Storage[C]//Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2012 14th International Sympo-sium on. IEEE, 2012: 430-435

[3] Neumann R, Taggeselle S, Dumke R, et al. Combining Query Performance with Data Integrity in the Cloud: A Hybrid Cloud Storage Framework to Enhance Data Access on the Windows Azure Platform[C]//Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on. IEEE, 2012: 518-525

[4] Ramanathan S, Goel S, Alagumalai S. Comparison of Cloud database: Amazon's SimpleDB and Google's Bigtable[C]//Recent Trends in Information Systems (ReTIS), 2011 International Conference on. IEEE, 2011: 165-168

[5] Wang H, Shea R, Wang F, et al. On the impact of virtualization on Dropbox-like cloud file storage/synchronization services[C]//Proceedings of the 2012 IEEE 20th International Workshop on Quality of Service. IEEE Press, 2012: 11

[6] Baidu PCS. http://developer.baidu.com/wiki/index.php?title=docs/pcs

[7] 金山 Kuaipan. http://www.kuaipan.cn/developers/document.htm

(上接第167页)

参 考 文 献

[1] Buneman P, Khanna S, Wang C T. Why and where: A characterization of data provenance[C]//Proceedings of the International Conference on Database Theory (ICDT). 2001

[2] Callahan S, Freire J, Santos E, et al. VisTrails: Visualization meets data management[C]//Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD). 2006

[3] Liu M, Taylor N E, Zhou W, et al. Recursive Computation of Regions and Connectivity in Networks[C]//ICDE. 2009

[4] Zhou W, Sherr M, Tao T, et al. Efficient querying and maintenance of network provenance at internet-scale[C]//SIGMOD. 2010

[5] Karvounarakis G, GIves Z, Tannen V. Querying data provenance [C]//SIGMOD'10 Proceedings of the 2010 International Conference on Management of Data. 2010: 951-962

[6] Green T J, Karvounarakis G, Taylor N E, et al. ORCHESTRA: Facilitating collaborative data sharing[C]//Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD). 2007

[7] Cohen-Boulakia S, Biton O, Cohen S, et al. Addressing the provenance challenge using zoom[J]. Concurrency and Computation: Practice and Experience, 2008, 20: 497-450

[8] Ikeda R, Park H, Widom J. Provenance for generalized map and reduce workflows[C]//Proceedings of Biennial Conference on Innovative Data System Research (CIDR). 2011

[9] System W, Altintas I, Barney O, et al. Provenance collection support in the kepler scientific workflow system[C]//Proceedings of the International Provenance and Annotation Workshop (IPAW). 2006

[10] Hasan R, Sion R, Winslett M. Preventing history forgery with secure provenance[J]. ACM Transactions on Storage (TOS), 2009, 5(4): 1-43

[11] Green T J, Karvounarakis G, Tannen V. Provenance semirings [C]//Proceedings of the ACM Symposium on Principles of Database Systems(PODS). 2007

[12] Network Simulator 3[OL]. http://www.nsnam.org/

[13] Liu X, Guo Z, Wang X, et al. D3S: Debugging Deployed Distributed Systems[C]//NSDI. 2008

[14] RapidNet[OL]. http://netdb.cis.upenn.edu/rapidnet/

[15] Loo B T, Condie T, Garofalakis M, et al. Declarative Networking [C]//CACM. 2009

[16] 万亚平，冯丹，欧阳利军，等. 一种适用于P2P存储系统的自反馈故障检测算法[J]. 计算机科学，2010，37(2)：48-52