构造服务质量感知的发布/订阅系统①

杨煜尧[1,2], 杨宇威[1,2], 张扶桑[1,2], 金蓓弘[1]

[1](中国科学院 软件研究所, 北京 100190)
[2](中国科学院大学, 北京 100190)

摘 要: 发布/订阅系统, 作为提供分布式事件检测的中间件, 可支持多种应用的开发. 考虑到应用提供的服务质量常常取决于发布/订阅系统的服务质量保障能力, 构建了一个服务质量感知的发布/订阅系统 Phoenix, 并从架构、可靠性保障机制、及时性保障机制等多方面来提升系统的服务质量. 因此, Phoenix 能成功应对通信链路故障、代理故障和集群抖动, 从而保障了应用的正常运行, 同时, 它能尽力而为地满足用户指定的及时性需求. 实验结果表明 Phoenix 的可靠性好, 能较好地支持用户的及时性需求.

关键词: 发布/订阅系统; 服务质量; 可靠性; 及时性

Building a Quality of Service Awareness Pub/Sub System

YANG Yu-Yao[1,2], YANG Yu-Wei[1,2], ZHANG Fu-Sang[1,2], JIN Bei-Hong[1]

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

Abstract: Pub/Sub systems are the middleware which provides the distributed event detection, they can support a variety of application development. Considering application of the quality of services provided often depend on the guarantee capability of quality of service (QoS) in the Pub/Sub systems, the paper builds a QoS aware Pub/Sub system named Phoenix, which improves the QoS from the architecture, the reliability guarantee mechanism and the timeliness guarantee mechanism. As a result, Phoenix can successfully deal with link failure, broker failure and cluster churn to guarantee that the applications can run normally. Moreover, it manages to satisfy the timeliness specified by the users with its best effort. The experimental results show that Phoenix is fairly reliable and supports the timeliness requirements of users well.

Key words: Pub/Sub systems; quality of service; reliability; timeliness

发布/订阅系统也被称为发布/订阅服务或事件分发中间件, 它通过松耦合的消息传递提供分布式的事件检测功能, 并为基于事件的分布式应用提供开发支持. 一个发布/订阅系统通常包含多个服务器(下文称代理), 用户可以向某个代理提交订阅以表达其所关心的事件要满足的条件, 同时, 事件及事件广告被发布到系统中的某个代理. 代理负责将事件与订阅进行匹配, 并将匹配成功的事件发送给感兴趣的用户.

随着应用领域的拓展和应用规模的扩大, 不少基于事件的分布式应用对非功能性需求如性能、可伸缩性、服务质量(QoS, Quality of Service)提出了更高的要求, 例如, 医疗健康应用有及时性需求, 举例而言, 监视器报告的表示患者生命体征不正常的事件必须要及时传递到有关人员; 而金融服务则需要有高可靠性保障, 以避免服务失效造成经济损失. 从发布/订阅系统的设计经验中可知, 系统内在的松耦合有助于提升可伸缩性, 而将发布/订阅系统构建在云平台之上, 可以充分利用云基础平台在资源分配调度上的可伸缩能力, 从而提升系统的可伸缩性, 但有些云平台(例如, Amazon EC2)支持以竞价方式获得计算实例, 也就是说, 云平台将根据供求关系和用户可接受的价格来自动开启和停止虚拟机. 这就需要云平台上的发布/订阅

① 基金项目:国家自然科学基金(61472408,61372182)
收稿时间:2015-04-13;收到修改稿时间:2015-06-03

系统能处理这种不稳定性, 从而保障应用的可靠性.

发布/订阅系统能提供的服务质量可以分为 5 类[1], 即事件传递的带宽资源、事件传递的时间延迟、事件传递语义、可靠性和消息排序方式. Shruti P. Mahambre 等人[2]分析了发布/订阅系统的多种可靠性, 并基于 P2P 覆盖网 Pastry 给出了查找用于事件路由的能满足用户指定可靠性的路径的方法. Christian Esposito 等人[3]则是从系统故障容忍和恢复的角度来提升发布/订阅系统的可靠性. 而 Miguel Matos 等人提出了 BRISA 方法[4], 该方法结合了传染病数据分发和基于树的数据分发策略, 以便同时保障数据分发的性能和可靠性. 为了相同的目的, Zhao 等人[5]提出了一种混合的系统架构, 即代理首先通过一个树型结构相连, 然后在其上构造一个 P2P 覆盖网. 而发布/订阅服务 BlueDove[6] 是将服务器组织成一个基于闲聊的一跳覆盖网络, 以提高系统的可伸缩性和可靠性.

发布/订阅系统的可靠性和及时性是最重要的 QoS 需求. 其中, 其中可靠性主要解决发布订阅中可能出现的通信故障、节点故障以及集群抖动等问题.

①通信故障: 链路的连通性故障, 会导致数据包经常丢失或完全无法通信.

②节点故障: 节点由于硬件或软件故障而失效.

③抖动故障: 节点或集群意外地加入或者离开.

及时性保障则为了解决具有差异化的及时性需求. 一方面, 不同的应用对于不同的事件具有不同的及时性要求; 另一方面, 同一应用下不同用户对于同一事件也有不同的及时性要求. 在当前系统资源的条件下尽量满足不同的及时性需求并达到全局的最优化, 是及时性保障需要解决的问题.

为此本文研究了相应的 QoS 保障技术, 并对原有的 OPS4Cloud[7]进行了重构, 实现了一个服务质量感知的发布/订阅系统 Phoenix. Phoenix 与原有系统一样基于云基础平台, 但采用了基于集群的体系架构, 以提高系统的性能和可伸缩性; 同时, 该系统针对通信链路故障、代理故障和集群抖动, 分别提供了冗余路径策略、基于复制的分类型代理恢复策略和集群加入和离开策略; 此外, Phoenix 允许用户指定及时性要求, 并基于性能感知提供相应的及时性保障机制. 可靠性实验结果表明 Phoenix 系统能够正确应对多种故障, 并采取正确的恢复策略, 从而实现系统整体的正常运行. 及时性实验结果表明在采取及时性保障机制后, 系统有效地提高了事件处理的及时程度.

下文按如下方式组织: 第二节给出了 Phoenix 系统概述, 第三节、第四节分别介绍系统的可靠性保障机制和及时性保障机制, 第五节是实验及结果分析, 最后是全文总结.

1 Phoenix系统概述

Phoenix 发布/订阅系统是原 OPS4Cloud 云平台发布/订阅系统的演进版本. 在原来的基础上增加了服务质量保障机制, 用以提供可靠性保障和及时性保障. Phoenix 系统构建在 XenServer[8]之上, 包含若干个集群(Cluster). 一个集群有且仅有一个路由主代理, 但它可包含一定数量的路由从代理、一个或者多个匹配主代理, 以及一定数量的服务于每个匹配主代理的匹配从代理. 图 1 是一个 Phoenix 实例, 用于说明 Phoenix 的架构组成.

在 Phoenix 中, 路由主代理按 Chord 环[9]组织, 也就是说, 这些代理以代理的 ClusterID 为主键值排列成 Chord 环, 并接收客户的广告、事件和订阅消息. 路由主代理以这些消息的事件类型作为主键, 根据 SHA-1 一致性哈希函数, 获得负责处理这些消息的路由主代理, 并将收到的消息路由到那些路由主代理. 这样, 系统的负载被分散到每个集群上, 而每个集群只负责处理一部分的广告、事件和订阅. Phoenix 为每个路由主代理配备了一定数量的路由从代理, 它们保存路由主代理上的路由信息副本, 同时负责路由主代理的故障检查与恢复. 而集群中的匹配主代理与该集群的路由主代理相连接, 接收来自路由主代理的广告、订阅和事件消息. 每个匹配主代理会维护一张路由表[10,11], 并通过查找路由表将对应的消息转发到集群内相应的匹配主代理上. 如果查询路由表后得出该消息由本代理负责时, 那么它将根据消息的类型来进行分类处理. 如果是广告和订阅消息, 则储存在本地的路由表和按事件空间组织的订阅森林中; 如果是事件消息, 则在本地进行匹配, 或者发给该匹配主代理的从代理进行匹配. 匹配从代理保存主代理上的路由表、广告和订阅副本, 对事件和订阅进行匹配, 它还负责监测匹配主代理的工作状态.

Phoenix 系统主要处理三类故障: (1)通信链路故障, 例如图 1 中 R1 和 R3 断连; (2)代理故障, 例如 M1 崩溃或宕机; (3)集群抖动, 例如 R1 所在集群离开系统,

它可以看做是代理故障的超集. 当云平台支持竞价型收费模式时, 集群抖动可能经常发生, 因此需要提供集群抖动的恢复策略, 使得集群加入和离开都不影响 Phoenix 系统的正常运行. 目前, Phoenix 系统提供这三类故障的检测和恢复机制.

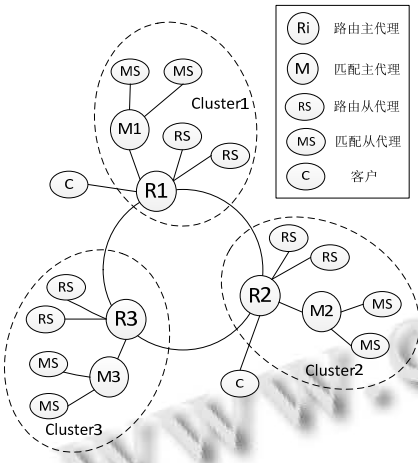

图 1 Phoenix 架构实例

2 可靠性保障机制

Phoenix 系统的可靠性保障机制是在系统新的自身的拓扑架构和系统特性的基础上进行设计与实现的. 为了提高可靠性保障的完备程度, Phoenix 可靠性保障机制的设计从通信、节点、集群三个方面进行切入和研究, 主要解决 Phoenix 发布/订阅系统中可能出现的通信故障、节点故障以及集群抖动.

考虑到通信链路故障、代理故障、集群抖动三种故障的检测有一定的关联性, Phoenix 给出了如下的检测策略: 首先, 路由主代理、路由从代理定期交换心跳消息, 并根据能否收到对方的心跳消息来判断对方是否出现故障. 匹配主代理、匹配从代理也按这种方式检测对方是否出现了故障. 其次, 当一个代理 P 在向另一个代理 Q 发送消息时发现它们之间的连接断开了, 这时需要区分 P 是否是想直接与 Q 建立连接, 若是(例如, P 是匹配主代理, Q 是 P 的匹配从代理), 那么连接不上意味着代理 Q 出现了故障; 另一方面, 如果代理 P 与代理 Q 可通过 Chord 环相连, 也就是 P、Q 都是路由主代理时, 那么 P 将激活有限步洪泛算法来检测是否发生了通信链路故障以及是否在 Chord 环上存在从源到目的地之间的冗余路径. 如果消息通过有限步洪泛能到达消息的目的地, 那么则意味着系统存在通信链路故障, 并保留洪泛过程中从源到目的地的路径(记为冗余路径), 否则, 则意味着代理 Q 发生了故障, 特别的, 如果 P 与 Q 的所有路由从代理也无法通信的话, 则意味着 Q 所在集群已经离开. 最后, 如果发现有新的路由主代理试图加入到 Chord 环中, 那么意味着有新的集群加入到 Phoenix 系统中.

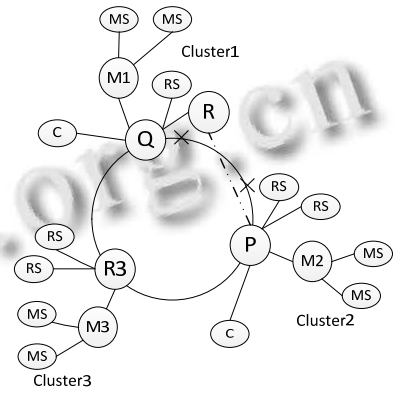

图 2 路由主节点恢复策略

通信链路故障的修复取决于是否找到上述的冗余路径. 如果找到冗余路径, 那么源节点将把消息发送给冗余路径中的第一个节点, 并在发送消息中包含冗余路径, 以指导接收节点如何转发该消息.

修复代理故障意味着需要采用新的代理去代替旧的代理以承担其之前的任务. 保存相关信息是修复代理故障的前提条件, 因此 Phoenix 系统定期将匹配主代理的包括本地路由表、广告池、订阅树结构和订阅对象等在内的信息持久化到 MongoDB. 同时, 采用法定数共识(Quorum consensus)方法[12]管理路由从代理和匹配从代理上的复制数据. 其次, Phoenix 系统为不同类型的代理故障提供了不同的修复策略.

限于篇幅, 本文仅给出路由主代理的修复策略. 如图 2 所示, 当 Chord 环上的某一路由主代理(设为 P)检测到其前驱(设为 Q)出现故障后, 它就开始将需要转发给 Q 的消息储存在本地的 bufferMessageList 里. 同时, Q 的下属从代理由于 Q 节点故障, 无法收到来自 Q 的心跳消息, 便开始启动霸道选举算法(Bully Algorithm)[12], 从路由从代理中选举出一个作为新的路由主代理(设为 R). 新当选的 R 还需要一系列后续操作才能承接起 Q 的职责, 具体内容如下:

1) R 修改自身的 profile 参数, 将自己的身份从路由从代理更换为路由主代理.

2) R 向管理虚拟机的守护进程发送请求, 申请一台新的虚拟节点, 并为其生成新的 profile 配置文件. 待新节点启动完成之后, R 需要将 profile 文件发送给启动完成的节点, 将其配置成新的路由从代理. R 将新配置完成的路由从代理加入到 slaveIdList 中, 以保证自己拥有足够的从代理数目, 从而继续提供可靠的服务.

3) R 向集群内所有的匹配主代理发送 HELLO 消息, 并通过 HELLO 消息建立连接. R 是通过之前作为路由从代理时同原路由主代理交换心跳信息得到集群内的匹配主代理信息(包括其标识符、IP 地址和端口号)的.

4) 集群内的匹配主代理收到 R 的 HELLO 消息后, 需要修改 profile 文件里的 chordBroker 相关信息, 将其改为新的路由主代理, 并回复 HELLO_REPLY 消息.

5) R 在收到所有的 HELLO_REPLY 消息后, 开始重新加入 Chord 环.

6) R 向其 Chord 环上的最近后继, 也就是 P, 发送 REJOIN 消息, 表明其已经代替原来的故障代理加入 Chord 环中.

7) P 在收到 REJOIN 消息后, 将之前缓存在 bufferMessageList 中的消息发送给 R.

8) R 需要优先处理掉收到的 bufferMessageList 中的消息.

至此, 系统恢复正常工作.

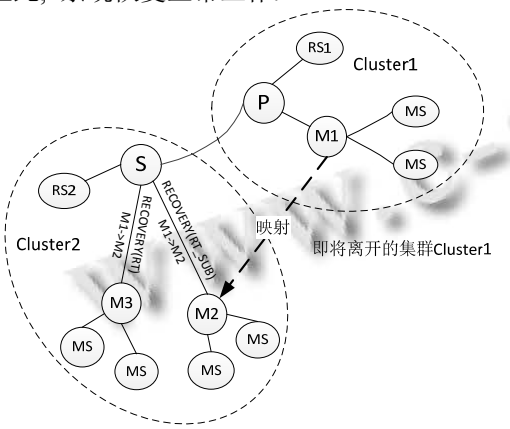

图 3 集群离开

同样由于受篇幅限制, 本文仅介绍集群离开后系统的恢复策略. 以图 3 为例, 当路由主代理 S 检测到它与其前驱节点 P 及其路由从代理 RS1 都无法建立连接时, 它认为 Cluster 1 离开, 并开始执行以下步骤:

1) 将原本由 Cluster1 负责处理的订阅和事件等消息缓存在本地的 bufferMessageList 中.

2) S 根据之前与 P 的心跳信息确定 Cluster1 包含的匹配主代理数目 N 和各个匹配主代理的标识符.

3) S 根据 N 和自己所在集群包含的匹配主代理数目 M 确定是否需要向虚拟机管理进程请求新的虚拟节点. 当 N>M 时, 将启动 k*(N-M)个新的虚拟节点(其中 k 为每个匹配主代理管理的匹配从代理数目), 在为它们生成配置文件后, 远程启动这些虚拟节点并配置成为本集群内的匹配主代理和匹配从代理; 若 N<=M, 则无需申请新的虚拟节点.

4) S 对本集群内的匹配主代理与 Cluster1 中的匹配主代理进行一一映射. 由(3)可知, S 所在集群 Cluster2 的匹配主代理数目将不少于离开的集群, 所以, Cluster1 中每个失效的匹配主代理都可以在 Cluster2 内找到一个映射代理. S 负责在匹配代理映射表 matchBrokerMap 中存入两个集群中需要相互映射的代理.

5) S 根据(4)确定的映射关系, 将 Cluster1 中的一个匹配主代理的标识符和映射表 matchBrokerMap 封装成 RECOVERY 消息发送给本集群中对应的匹配主代理. 这里的 RECOVERY 消息分为两类, 一类是 RECOVERY(RT_SUB)消息, 另一类是 RECOVERY (RT)消息. 根据(3)区分的两种情况, 若 N<M, S 向其所在集群中需要对离开集群的订阅进行恢复的匹配主代理发送 RECOVERY(RT_SUB)消息, 而对其他匹配主代理发送 RECOVERY(RT)消息.

6) Cluster2 的匹配主代理在收到 RECOVERY 消息后, 将根据 RECOVERY 中包含的对应映射代理的信息去恢复已持久化存储的路由表和广告等数据. 如果收到的是 RECOVERY(RT_SUB)消息, 则该匹配主代理还需要在内存中构建订阅树结构, 同时恢复相应的订阅. 当 Cluster2 的匹配主代理在完成恢复操作后, 向 S 回复 SUCCESS 消息.

7) 当 S 收到所有的匹配主代理发送的 SUCCESS 消息后, 开始将保存在本地 bufferMessageList 中缓存的消息随机地发送给本集群内的匹配主代理, 并由它们进行处理.

3 及时性保障机制

时性保障是为了解决不同客户对于不同事件响应时间的要求, 从而达到全局的及时性满意程度最大化.

为了有效地提高系统的及时性保障, Phoenix 系统加入了集群间和集群内两种即时性保障机制, 并给出了对于系统及时性的评估方法.

考虑到不同用户对于不同的事件有不同的响应时间要求, Phoenix 系统提供了 API, 允许用户为特定事件类型指定相应的及时性要求. 及时性要求用事件的延迟容忍度 latencyLevel 刻画, 其取值范围是 1 到 L($L \in$ N, L>=1). 对于某一特定事件类型的事件, 用户为其设定的延迟容忍度越小, 表明用户对该类型事件的及时性要求越高. 在 Phoenix 系统中, 每个路由主代理(如图 1 中的 R1、R2、R3)都维护一个延迟容忍度映射表 latencyMap: Map<eventType: String, latency: Latency>, 其中, latency 包含两个属性 latencyLevelSum 和 counter. 当用户为某个 eventType 指定 latencyLevel 并提交后, 路由主代理将更新 latencyMap: 将用户关注的 eventType 下的 latencyLevalSum 加上新收到的延迟容忍度属性 latencyLevel 的值, 并将 counter 的值加 1. 这样, 通过 latencyLevalSum 和 counter 的比值就可以获得用户对 eventType 类型的事件的平均延迟容忍程度.

由主代理将根据事件的平均延迟容忍度来确定该事件的优先级, 并基于此优先级来将事件消息插入消息转发队列, 从而使得具有较低平均延迟容忍度的事件可以优先得到转发, 从而提升集群间路由的及时性. 本文把上述策略称为集群间的及时性保障策略.

另一方面, 为了改善集群内事件处理的及时性, 本文提出了基于性能感知的集群内及时性保障策略, 即, 每个匹配主代理和从代理根据数据统计来预测本地的事件处理平均时间, 而且, 匹配从代理还将该预测时间值报告给匹配主代理, 这样, 匹配主代理在收到事件消息后, 将根据事件类型选出具有最短事件处理预测时间的匹配代理. 若选出的代理为匹配主代理自身, 那么在本地处理该事件消息; 否则, 将该事件消息发送给相应的匹配从代理.

这里, Phoenix 按以下方式预测上述策略中的平均事件处理时间:

1) 每个匹配代理(包括主代理和从代理)周期性统计在过去的一段时间 t 内到达本地的事件数目 NA 以及本地完成匹配计算的事件消息的数目 NC, 并计算在本地的事件消息的平均到达率 a 和平均匹配率 β, 即 $a = NA/t$, $\beta = NC/t$.

2) 设每个匹配代理在 t' 时刻统计的消息队列长度为 $q(t')$, 那么, 需要预测的本地 t 时刻的消息队列长度 $q(t)$ 可按下式计算: $q(t) = q(t') + (a-b)*(t-t')$.

3) 对于事件消息而言, 匹配代理处理事件消息的时间 $T_{processing}$ 分为两部分, 一部分为在消息队列中的等待时间 $T_{waiting}$, 一部分为消息进行匹配计算的时间 $T_{computing}$.

对于消息队列里的第 i 个消息 ei, 它的等待时间 $T_{waiting}(ei) = (i-1)/\beta$, 它的匹配计算时间 $T_{waiting}(ei) = 1/\beta$. 因此, 消息 ei 的处理时间为 $T_{processing}(ei) = 1/\beta$.

这样, 基于预测出的在时刻 t 的消息队列长度 $q(t)$, 可以得出在时刻 t, 对于消息队列中的所有事件消息(记作: $e1$, $e2$, …, $eq(t)$), 匹配代理的平均事件处理时间 $T_{processing}(t)$ 可按下式计算:

$$T_{processing}(t) = \sum_{i=1}^{q(t)} T_{processing}(ei)/q(t) = \frac{(q(t)+1)}{2\beta} = \frac{(q(t')+(\alpha-\beta)*(t-t')+1)}{2\beta} \quad (1)$$

根据公式(1)可以得出, 若匹配代理能够计算出自身事件消息的平均到达率 α 和平均匹配率 β, 并在时刻 t' 检测得到消息队列长度 $q(t')$, 则可以预测出在时刻 t, 本地的平均事件处理时间 $T_{processing}(t)$.

4 实验评估

本文的实验环境包括了两台 Dell PowerEdge T610 服务器, 它们均配置有 4 颗双核处理器和 12GB 内存. 同时, 两台服务器安装了 XenServer(v5.5), 并在每个 XenServer 中安装了多个以 1 核 CPU 和 1GB 主存为基本配置的虚拟机, 并让所有的虚拟机工作在 CentOS Linux 5.4 下. 此外, 本文还设计实现了一个可视化监控工具, 用于观察 Phoenix 系统拓扑结构和通信链路状况.

1) 可靠性保障实验

在可靠性保障实验中, 本文模拟了一系列故障, 包括链路故障、代理故障和集群抖动, 并记录了系统的恢复时间. 通过比较故障前后的系统拓扑和通信链路状况, 实验结果显示 Phoenix 系统能从故障状态恢复正常. 以链路故障实验为例, 图 4-图 8 分别显示了发现链路故障、发起洪泛算法、找到冗余路径和沿着冗余路径发送消息的情景, 它们都是来自可视化监控工具的截图. 而在代理故障实验中, 通过人为杀掉代理

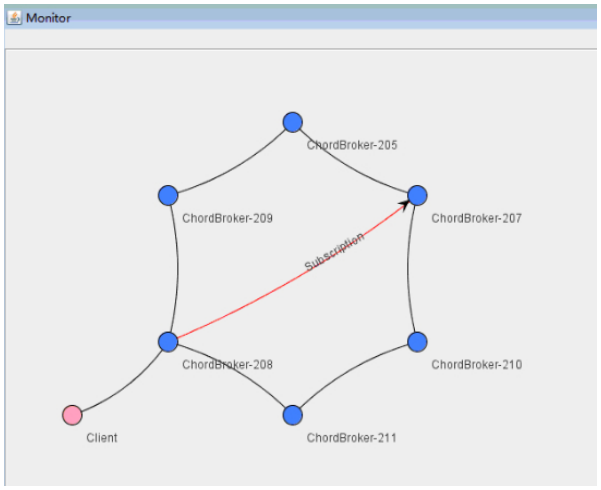

图 4 通信链路出现故障

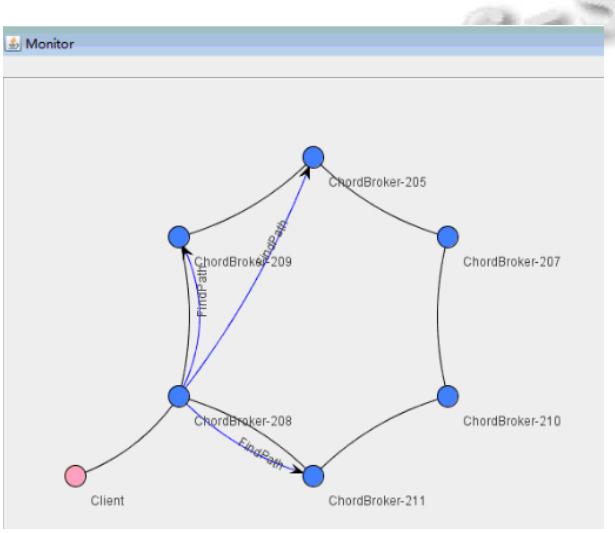

图 5 洪泛寻找新的通信路径

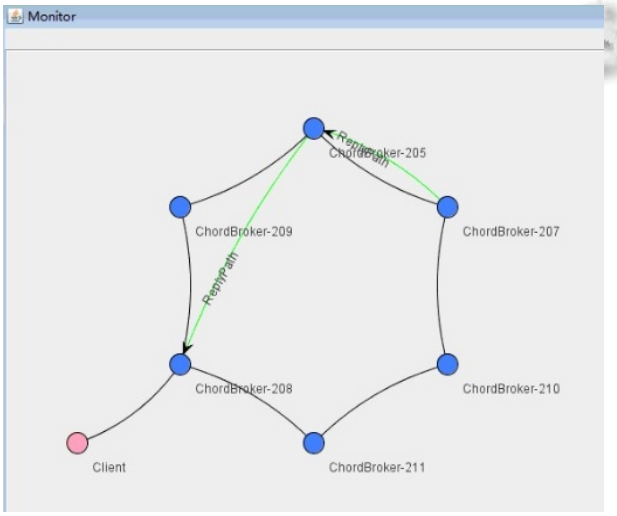

图 6 找到新的通信路径

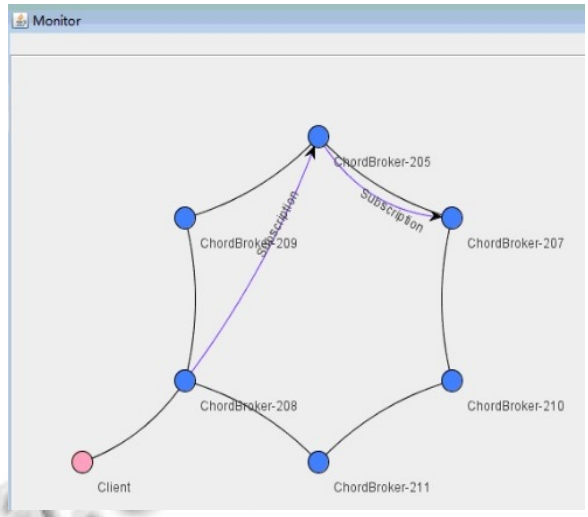

图 7 找到新的通信路径

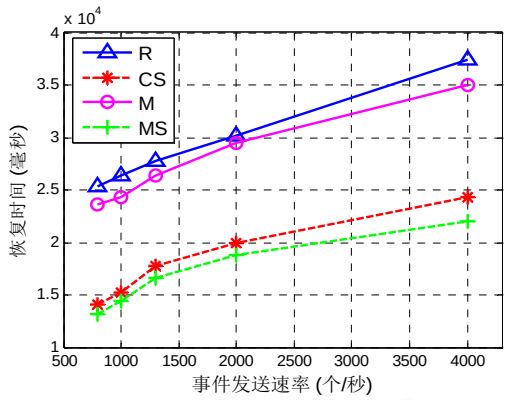

图 8 四种代理的故障检测与恢复时间

上的进程来模拟代理故障的出现, 并在不同的事件发送速率下, 统计故障的检测和恢复时间, 这段时间包括: 故障检测时间, 恢复过程中所有消息(如 profile 消息、rejoin 消息、hello 消息等)的生成和传递时间, 持久化数据恢复时间, 代理的配置时间和连接重新建立时间. 其中路由主代理和匹配主代理还需要包含执行霸道选举的时间. 图 7 给出了不同类型代理的故障检测和恢复时间. 例如在事件发送速率为 2000 个/秒时, 路由主代理、路由从代理、匹配主代理、匹配从代理的平均恢复时间分别为 30124 毫秒、19930 毫秒、29467 毫秒、18829 毫秒.

2) 及时性保障实验

在及时性保障实验中, 系统部署了如图 1 所示的 3 个集群. 在每个集群中, 部署 1 个路由主代理, 每个路由主代理对应部署 2 个路由从代理; 部署 1 个匹配主代理, 每个匹配主代理对应部署 2 个匹配从代理; 部

署 2 个客户, 每个客户与一个路由主代理相连. 这里, 路由主代理和匹配主代理分别被部署在单个虚拟机上, 而一个路由从代理和一个匹配从代理被部署在同一台虚拟机上.

将集群内和集群间及时性保障策略进行了组合, 得到 4 种及时性保障机制, 如表 1 所示.

表 1　四种及时性保障机制

机制编号	机制名称	集群间及时性保障策略	集群内及时性保障策略
1	无保障	×	×
2	内部保障	×	✓
3	外部保障	✓	×
4	双重保障	✓	✓

在第一组实验中, 将客户提交的订阅总量设为 200000 个, 调整客户发送事件的速率, 观察四种机制对事件消息的平均响应时间. 由图 8 所示的实验结果可以看出, 采用了及时性保障机制后, 系统对事件消息的平均响应时间降低. 具体而言, 在不同的事件发送速率下, 采用双重保障和内部保障机制, 得到的系统对事件消息的平均响应时间较为接近, 且均低于采用另外两种机制得到的平均响应时间. 另一方面, 随着事件发送速率的提高, 对于每一种机制, 系统对事件消息的平均响应时间均有所增加, 但增长的幅度不大. 可以反映出在四种机制下, Phoenix 系统均具有较好的可伸缩性.

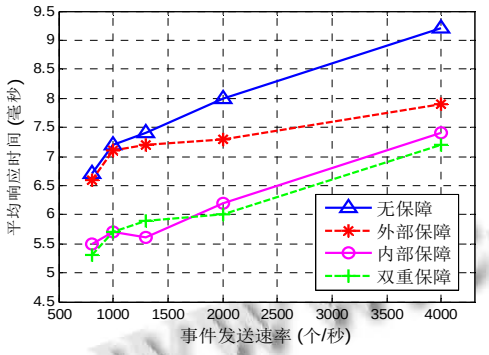

图 9　不同事件发送速率下的平均响应时间

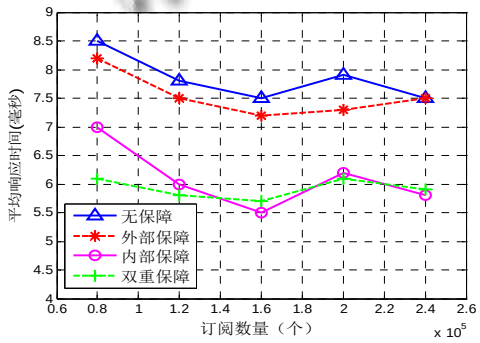

图 10　不同订阅数量下的平均响应时间

在第二组实验中, 将事件发送速率设为 2000 个/秒, 调整客户提交的订阅数目, 观察四种机制对事件消息的平均响应时间. 由图 9 所示的实验结果可以看出, 采用了及时性保障机制后, 系统对事件消息的平均响应时间降低了. 具体而言, 采用双重保障和内部保障机制, 得到的系统对事件消息的平均响应时间较为接近, 且均低于采用另外两种机制得到的平均响应时间. 另一方面, 随着订阅数量的增加, 对于每一种机制, 系统对事件消息的平均响应时间均在一个较小的范围内波动, 没有明显的增加. 可以反映出在四种机制下, Phoenix 系统均具有较好的可伸缩性.

5　总结

为了改善发布/订阅系统的服务质量, 本文对原有的云平台上的 OPS4Cloud 进行了重构, 形成了发布/订阅系统 Phoenix. 与原有系统相比, Phoenix 系统采用了集群结构, 特别提供了系统的可靠性保障, 使得系统能有效应对链路故障、代理故障和集群抖动; Phoenix 系统允许用户指定应用所需的及时性要求, 并提供了集群间和集群内的及时性保障机制. 实验评估表明 Phoenix 系统有很好的可伸缩性和鲁棒性, 能较好地支持用户的及时性需求.

参考文献

1 Mahambre SP, Kumar MSD, Bellur U. A taxonomy of QoS-aware, adaptive event-dissemination middleware. Internet Computing, IEEE, 2007, 11(4): 35–44.

2 Mahambre SP, Bellur U. Reliable routing of event notifications over P2P overlay routing substrate in event based middleware. 2007 IEEE International Parallel and Distributed Processing Symposium. Long Beach, USA. 2007. 468.

3 Esposito C, Cotroneo D, Gokhale A. Reliable publish/subscribe middleware for time-sensitive internet-scale applications. Nashville, TN, USA. DEBS'09. 2009.

4 Matos M, Schiavoni V, Felber P, Oliveira R, Riviere E. BRISA: combining efficiency and reliability in epidemic data dissemination. 2012 IEEE 26th International Parallel and Distributed Processing Symposium. Shanghai, China. 2012.

983–994.

5 Zhao YX, Wu J. Building a reliable and high-performance content-based publish/subscribe system. Journal of Parallel and Distributed Computing, 2013, 73(4): 371–382.

6 Li M, Ye F, Kim MK, Chen H, Lei H. A scalable and elastic publish/subscribe service. 2011 IEEE International Parallel & Distributed Processing Symposium. Anchorage, Alaska, USA. 2011. 1254–1265.

7 Fang WJ, Jin BH, Zhang B, Yang YW, Qin ZY. Design and evaluation of a Pub/Sub service in the cloud. 2011 International Conference on Cloud and Service Computing. Hong Kong, China. 2011. 32–39.

8 Barham P, Dragovic B, Fraser K, et al. Xen and the art of virtualization. ACM Symposium on Operating Systems Principles. Bolton Landing, NY, USA. 2003. 164–177.

9 Stoica I, Morris R, Karger D, et al. Chord: a scalable peer-to-peer lookup service for internet applications. ACM SIGCOMM. San Diego, California, USA. 2001, 31(4): 149–160

10 Li S, Qian YF, Yang YW, Jin BH. Adaptive channel-based routing in content-based Pub/Sub systems. 2013 IEEE International Conference on Internet of Things. Beijing, China. 2013. 504–511.

11 Carzaniga A, Rosenblum DS, Wolf AL. Design and evaluation of a wide-area event notification service. ACM Trans. on Computer Systems, 2001, 19(3): 332–383.

12 Coulouris G, Dollimore J, Kindberg T, Blair G. Distributed Systems: Concepts and Design. 5th Edition, 2012.