一种基于内容的数据分发网络及算法

朱昭萌 张功萱 张永平 郭 箭 张 巍
(南京理工大学计算机科学与技术学院 南京 210094)

摘 要 物联网中通常需要对海量传感数据进行有目的的聚合和分发。设计中通常使用集中式的大规模存储系统管理海量物联数据,合适的数据分发机制对这样的存储系统必不可少。提出一种基于内容的数据分发网络设计。该设计可以和大规模存储中节点网络相结合,在保证存储过程不受影响的前提下收集空闲计算资源,高效完成基于内容的数据分发工作。受虚拟化思想的启发,设计引入"工作者"和"功能块"角色,通过动态调节每个节点上"工作者"的数量实现网络中节点的负载自动调节,同时通过动态装载或切换"功能块"实现不同任务之间计算资源的动态调配。还具体给出了一种基于布隆过滤器的分布式基于内容的数据分发算法。该算法分布于上述数据分发网络,同时通过使用布隆过滤器表示对象及其属性所满足的约束集合,消除了大量冗余计算,可以并行、高效地完成数据分发工作。

关键词 基于内容分发,分布式,发布订阅,布隆表达式
中图法分类号 TP311 **文献标识码** A

Design of Content-based Data Forwarding Network and Algorithm

ZHU Zhao-meng ZHANG Gong-xuan ZHANG Yong-ping GUO Jian ZHANG Wei
(Department of Computer Science and Technology, Nanjing University of Science and Technology, Nanjing 210094, China)

Abstract The data in the Internet of Things needs to be aggregated and disseminated effectively. There are always central massive storage to manage the numeric data produced by sensors. Proper data forwarding mechanism is necessary for such a system. We proposed a design of content-based data forwarding network. This network can combined with the node network of massive storage, that is, utilizing the scattered computing resources of each node in a massive storage network to take up the challenging task of dissemination of a great amount of data. Inspired by the system virtualization, we introduced "workers" and "function bricks" to dynamically balance the loads of different nodes and tasks between nodes in the network automatically. We also proposed a distributed content-based publish/subscribe algorithm using Bloom filter. By efficiently using Bloom filter to represent the object as well as the set of constrains which is satisfied by the object, a large number of redundant computing can be eliminated to sure the algorithm's efficiency.

Keywords Content-based forwarding, Distributed computing, Publish/subscribe, Bloom filter

1 概述

通过集中式的大规模存储设施,物联网中传感器采集到的大量信息可以被综合处理、分析和归类,并分发给任何感兴趣的使用者,而使用者不必和传感器网络的制造、搭建者直接联系。高效实现数据的聚合和分发机制需要存储系统提供合适的数据接口。最常用的数据接口是查询,这种方法被现有的各种存储系统广泛支持,却有一个严重的缺点:查询过程必须由用户主动发起。但在物联网应用中,经常会出现以下问题:1)使用者需要接收某些数据,但无法事先得知数据的到达时刻;2)某些紧急数据不能等待查询才传递给使用者,而是一旦出现就要尽快传递给使用者;3)连续性的数据应该被自动且连续地传递给使用者,而不需要用户反复查询。

发布/订阅(Publish/Subscribe, pub/sub)机制实现了数据(或消息)生产者和消费者之间的解耦,使数据(或消息)的生产者和消费者都不需要知道对方的具体信息。订阅者通常只对特定的数据感兴趣,并且只接收感兴趣的数据。在基于内容的数据订阅/发布模型中,精确符合订阅者的订阅条件的数据会被主动推送给订阅者[1]。发布/订阅模型可以实现大量物联数据的有效分发。

现有大规模存储系统设计允许数千或者更多具有一定计算资源的存储节点构成网络[2,3]。另一方面,由于需要分发的数据量巨大,因此需要构建数据分发网络分布处理海量数据的分发。如果将这两个网络合并,不仅可以降低系统硬件投入,还会减少不必要的网络间流量,提高存储系统的性能。

面向大数据量的分布式基于内容的发布订阅算法及分发网络在多篇论文中已有论述[4-6]。但直接将其和支撑物联网的大规模存储系统合并会造成:1)由于物联网中传感器种类

到稿日期:2012-06-23 返修日期:2012-09-30 本文受江苏省973项目(BK2011022),国家自然科学重点基金(61170035),南京理工大学重点基金(2011YBXM18)资助。
朱昭萌(1988-),男,博士生,主要研究方向为分布式计算、云计算,E-mail:zhaomeng.zhu@gmail.com;张功萱(1961-),男,博士,教授,主要研究方向为网络与分布式计算、Web服务。

丰富，数据的动态性很强，不同种类数据到达的流量等随时间分布不均匀，简单的分布无法应对复杂多变的物联网环境。2）如果直接将数据分发网络和节点存储网络合并，没有合适的节点负载和资源调节机制，数据分发可能会影响存储系统的正常运作。

本文提出一种将基于内容的发布订阅网络和已有大规模存储系统中节点网络相结合的方案。该方案提供了负载调节和资源调配的功能，通过收集利用大规模存储系统节点网络中不同计算节点上零散的计算资源，来高效完成物联数据基于内容的分发任务，在节省系统流量的同时降低了系统的总成本。

本文第 2 节给出基于内容发布/订阅系统的相关定义和术语；第 3 节阐述设计目标；第 4 节首先给出数据分发的基本流程和总体框架，然后对该方案提供的各种机制和为实现设计目标所采取的算法分别进行了详细阐述和讨论；第 5 节给出一种具体的基于布隆过滤器的数据分发算法；最后对全文进行总结。

2 定义

一个“对象”包括数据和表述数据特征的一系列键值对属性。例如一个摄像头采集的图片可能会附加一些数据来标识采集地点和采集时间等信息。此类描述性的信息被称为“元信息”，其中每个键被称为“属性”。

“订阅”是事先注册在系统中的一系列条件。一个“订阅”包括一系列断言组成的析取范式以及一个“目标地址”。例如，对于以下给出的一个订阅的例子：

{{“Location”=“A5” ∧ “Time” > 20 ∨ “Type”=“image” ∧ “Producer” IN {“Camera-30A4”,“Camera-30A5”}}, address=user0@example. com}

其含义为：每个满足{“Location”=“A5” ∧ “Time” > 20}或{“Type” = “image” ∧ “Producer” in {“Camera-30A4”, “Camera-30A5”}}的对象将被推送给“user0@example. com”。

一个“断言”包括一个“约束”和一个属性名称。例如[“Location”=“A5”]是一个包含约束[“=”,“A5”]和属性名称“Location”的断言。“约束”是测试数据时的最小单元，包括一个值和一个“操作符”。不同数据类型有不同的操作符，但每个操作符都是双目的，并生成一个布尔值。

后文将多次出现“断言”和“约束”，这里对其进行特别说明。“约束”是针对属性的，而“断言”是针对对象的。因此“某属性符合约束”即表示某对象符合一个包含该属性的断言。后文为叙述方便，将在针对特定属性的语境中直接使用“约束”来表达一个包含该属性的断言。

本文只考虑两种数据类型：“数字”和“字符串”。因为布尔型可以被数字或字符串标识，本文并不单独考虑布尔型。对于数字类型，定义“大于/<”、“小于/>”和“等于/=”；而对于字符串类型，考虑“等于/=”、“前缀/PF”和“包含与/IN”。其中“包含与”操作检查字符串是否属于一个给定的字符串集合。此外，在本文的方案中，订阅格式虽然是一个合取范式，但可以被拆分为多个具有相同目标地址的析取式。为方便描述，本文只讨论订阅条件中仅包含单个析取式的订阅。

3 设计目标

大规模存储系统由大量节点构成。将数据分发网络直接整合到存储系统中可以 1）避免数据在两个系统之间频繁交换，以降低通信量；2）充分利用存储系统中已有硬件资源，降低硬件投入，从而降低整个系统的总拥有成本（TCO）[5]。但大规模存储系统为方案设计提出一些不同的设计目标。

大规模存储系统中，整个系统的性能通常更加依赖于数据的复制、路由、移动等存储相关过程，数据分发过程必须非常小心，否则会影响正常的存储过程。换言之，数据分发方案需要具有节点负载自动调节能力，这样可以在存储过程中需要较多计算资源时自动释放计算资源，而不影响存储系统性能。同时，存储系统中的节点数目虽然众多，但每个计算能力通常并不强大。为了充分利用每个节点有限的硬件资源，负载调节能力的调节粒度必须尽可能细。

另外，物联网通常包括大量不同种类的传感器，各传感器产生数据的时间、种类和特点也各不相同，加上不同数据的分发需求千差万别，导致分发机制要处理的数据流量、类别等难以预计。需要设计提供健壮的自调节能力使得系统可以根据需求动态调配不同的分发任务所需的资源，从而高效完成分发任务。

4 方案设计

4.1 数据分发基本流程

对象到达存储系统时会被分配一个全局唯一的标识（ID）。分发过程只使用对象的元信息和 ID，流程结束时，所有匹配成功的订阅中的目标地址将会收到该 ID。

在本文的方案中，参与整个分发过程的角色包括一个协调者、多个测试工作者以及一个匹配工作者，基本流程如图 1 所示。

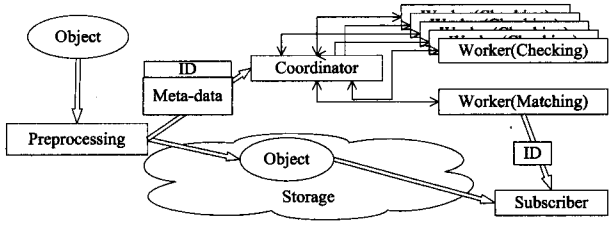


图 1 基本流程

1. 对象到达存储系统后，ID 和元信息被发送给协调者，同时对象交给存储系统存储；

2. 协调者将元信息分割为不同的属性和值，并分配不同的测试工作者完成测试工作；

3. 测试工作者检查并计算出一个包含该属性值所满足所有约束的集合，测试工作者完成任务后，缓存结果并通知协调者；

4. 全部工作者完成任务后，协调者指定一个匹配工作者完成匹配工作；

5. 匹配工作者从不同的测试工作者收集计算出的集合，在已注册的订阅信息中查找相关匹配，并将对象 ID 推送给所有匹配成功的订阅者。

4.2 详细设计和相关机制

为叙述方便，下文将一次“找出某个属性所满足的所有约束集合”的操作称为一个“任务”。

4.2.1 工作者

工作者是一个一次可以容纳一个任务执行的容器。现有

商用硬件通常包含多个独立计算核心，将一个物理节点划分成多个工作者可以更有效地利用计算资源。但设计中物理节点上容纳工作者的数量和计算核心数之间并没有严格的对应关系，因为还要考虑到不占用计算资源的 I/O 操作和需要保证计算资源的存储过程的存在。一个节点上可用工作者的数量可以在一定程度上反映该节点可用计算资源的多寡。工作者可以根据需要被动态创建或终止，因此工作者的存在为节点负载调节提供了机制。

4.2.2 功能块

在每个工作者中，完成实际任务的是功能块，而工作者只提供功能块运行时的环境。

工作者和功能块的概念一定程度是受系统虚拟化[7,8]和云计算中基础设施即服务(Infrastructure as a Service)[9,10]概念的启发。工作者为功能块提供执行时的虚拟环境。功能块是执行具体任务的代码和数据的组合。工作者执行的任务种类由其中容纳的功能块种类决定。测试功能块负责对某一个特定属性的值进行测试，找出其所有满足的约束。匹配功能块收集测试功能块的结果、对结果进行整合并最终查找出所有满足条件订阅的目标地址。图 2 说明了物理节点、工作者和功能块之间的关系。

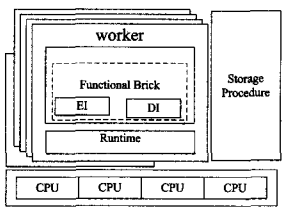


图 2 工作者与功能块

本文设计中，功能块由“可执行镜像”(Executable Image，简称 EI)提供的代码和“数据镜像”(Data Image，简称 DI)提供的数据组合而成。对测试功能块来说，EI 中实现针对特定数据类型的测试，DI 则存储特定数据结构，比如一棵包含所有“producer”属性相关约束中出现字符串的三叉树。通过动态组合不同的 EI 和 DI，可以获得特定功能的测试功能块。例如，从实现字符串相关测试的 DI 启动，并附加“name”属性相关 EI 的测试功能块，能够检查所有值为字符串的“name”属性相关约束。而对于匹配功能块，EI 的功能是收集测试功能块结果并查找出所有符合条件订阅的目标地址，DI 则包含所有订阅相关信息(实际上只包含订阅对应的布隆过滤器和相应目标地址，这些信息对匹配算法来说已经足够。基于布隆过滤器的具体数据分发算法将在下节详述)。

一个工作者装载的功能块并不是固定的，工作者可以动态装载不同的功能块，从而完成角色的转变。这种机制允许整个网络中各种功能块的比例动态变化，即处理不同任务的能力动态变化，为实现不同任务之间计算资源的动态调节提供了机制。

所有工作者启动时都是空工作者，即没有装载任何功能块的工作者。增加一个空工作者意味着节点认为有资源同时可以执行一项新任务，但将具体资源分配工作延后到实际需要时，会使得节点负载调节机制和资源调配机制相独立。

4.2.3 协调者

协调者负责管理并分配任务给不同的工作者。测试工作者按所能处理的属性名称组织成多个空闲队列。匹配工作者和空工作者也分别按队列维护。协调者还可以通过“CHANGE”指令要求一个工作者装载指定功能块。在分配任务或要求工作者装载新功能块后，工作者将从对应队列中删除，工作者完成任务或装载/切换功能块完成后将给协调者发送一条“ADD”信息，其中包括工作者当前装载的功能块标识。

考虑到系统的灵活性，协调者可以在任何时刻加入，或停止并在其他节点上重新启动，工作者也可以随时加入和离开网络。为实现协调者和工作者的动态协作，给出如下协议：

1. 协调者和工作者在启动时都会广播一个数据包通知网络中所有节点自己的存在，称之为“CRY 包”。

2. 协调者收到工作者的 CRY 包后，会将该工作者加入自己相应的队列，同时再广播一个 CRY 包。之所以在这里仍使用广播而不是单播，是因为 UDP 广播可能会丢包，人为地增加一些消息可以保证所有工作者都会最终在某一时刻被加入网络。

3. 工作者在收到协调者的 CRY 包之后，会用协调者的信息更新现有协调者信息。协调者会在 CRY 包中附带一个随机字段，功能块通过该字段判断这个协调者是否是一个新的协调者。

4.3 负载调节、资源调配、容错及相关讨论

4.3.1 节点负载调节

节点的负载调节由所在节点负责，节点会监视自己的资源利用率，若资源利用率超过某一个上限阈值，守护进程会终止所有空闲工作者以防止被分配更多的计算任务；若利用率低于某个下限阈值，守护进程会启动一些新的空工作者。

一个工作者中的任务实际上划分了算法中可分布进行运算的最小单元。元信息中不同的属性可以独立地进行测试而不会互相依赖。而对同一属性的测试则不应被分开，这是因为约束之间通常具有包含关系，很多运算是不必要的。例如，满足{<,5}约束的属性值，一定也满足约束{<,10}。因此这种以工作者为单位的调节机制提供了最细粒度的负载调节能力。

4.3.2 计算资源动态调配

计算资源的动态调配能力指系统中某种类型任务的数量增加时可以获得更多的系统资源。例如当某种类型数据大量到达时，负责对应属性相关测试工作者的数量也需要增加。为实现这种资源动态调节，协调者在分配任务时若发现当前没有对应的工作者，则会：

1. 若系统中还有空工作者，则命令一个空工作者装载对应任务的功能块；

2. 若系统中已无空工作者，则从目前最长的空闲队列中取出一个工作者将之切换到当前任务；

3. 若当前系统中已无任何空闲工作者，说明系统已满负荷，再增加任务会影响到正常存储过程，因此什么都不做；

4. 缓存任务，等待下一个可用工作者，工作者装载功能块完成会主动通知协调者。

4.3.3 工作者失效处理

复杂系统中硬件或软件失效在所难免，需要提供合适的容错机制。这里仅考虑数量较多的工作者失效情况。

由于工作者可以在自己变为可用时主动报告状态，协调

者只需要鉴别出已失效工作者并将之从队列中删除即可。工作者的失效主要分为两种：

1. 任务分发给工作者之后工作者失效，不能完成接收到的任务；

2. 工作者在空闲时失效，即工作者虽然仍在协调者的空闲队列中，但已失去响应。

对于前一种情况，由于每种工作者完成任务相同，协调者可以统计完成任务的平均时间。如果协调者发现一个工作者在超出平均时间很长时间内未有回应，则认为节点已经失效，会将任务重新分配给其他工作者。而对后一种情况，在分配任务到工作者之前，协调者和工作者之间会握手进行确认，若失败，则直接删除该工作者，再分配任务给列表中的下一个工作者。

而如果工作者由于网络中断等原因被协调者判断为已失效，该工作者不会再收到任何任务。这种工作者不占用计算资源，因此并不需要立即被清理掉，所在的物理节点负责定期清理那些长期没有任务的工作者即可。

5 基于布隆过滤器的测试和匹配算法

5.1 布隆过滤器

布隆过滤器[11]是一种概率性数据结构，通常用来测试一个元素是否在集合中。它的空间效率和时间效率都远远超过一般的算法，但存在一定的误识别率，即将不在集合中的元素判断为在集合中。布隆过滤器用一个长度为 m 的位数组来表示一个集合，初始时数组的所有元素都为 0。布隆过滤器通过 k 个哈希函数将一个元素映射到数组中的 k 个位置。添加元素时，将这 k 个位置都置为 1。检查一个元素是否在集合中时，检查这 k 个位置是否全部为 1。若全部为 1，则说明元素很有可能在集合中，否则元素一定不在集合中。布隆过滤器中插入和检测运算的时间复杂度都是 O(1)。在允许一定误算率的情况下，布隆过滤器具有很高的时间和空间效率。此外，布隆表达式还可以通过 OR 位操作和 AND 位操作快速地实现集合的并和交。

本文给出的方案实质上是由测试工作者求出对象每个属性满足的所有约束的集合，再由匹配工作者求出这些集合的并集并与已注册的订阅进行匹配。一个订阅也可以看成一个断言的集合，匹配一个对象和一个订阅，即是检查一个订阅的断言集合是否包含于对象满足的断言集合。布隆过滤器可以快速地对集合进行运算，因此本算法使用布隆过滤器来表示集合，即：

1. 每个测试功能块产生的结果是一个表示某属性满足所有约束集合的布隆过滤器；

2. 每个订阅的条件看成断言的集合，用一个布隆过滤器来表示；

3. 每个断言对应一个只包含一个元素集合的布隆过滤器。

布隆过滤器的问题是有一定的误判率，但我们认为这些误判不会造成严重的影响，因为：

1. 大规模存储系统中存在少量分发错误是允许的，订阅者在接受时进行简单判断即可。

2. 判断数据匹配订阅的过程是判断一个集合是否是另一个集合的子集，而不只是一个元素是否属于一个集合，这大大降低了误判效率。同时考虑到实际应用中不同的订阅通常针对不同的数据，不同的对象也拥有不同的属性，订阅的集合之间以及不同对象满足断言的集合之间都相对分离，发生误判的概率微乎其微。

3. 如果不能容忍任何的误判，可以在计算断言位置时检测并避免冲突，计算断言位置的运算实际上只在新的断言注册到系统中时做一次，其代价是能够容忍的。

5.2 数字类型测试

对于数字类型的属性。下面给出一个快速测试算法。

对于每个属性，算法维护一个 K 行 4 列的表，其中 K 是这个属性相关的所有约束的总数。表的第 1 列记录所有约束中出现过的值，第 2 列到第 4 列各对应一种运算。这样表中第 2 列到第 4 列中的每一个位置都对应一个约束，我们给每一个位置填充一个布隆过滤器。设(i,op)表示表格中 op 运算对应的列中的第 i 个元素，则(i,op)位置对应一个约束$\{op,k\}$，其中 k 为表格中第 1 列第 i 行的值。令 $F(i,op)$表示(i,op)位置的值，$f(i,op)$表示(i,op)位置所表示的约束对应的布隆过滤器(如果约束不存在则对应一个全 0 的布隆过滤器)，则表格中的值满足如下关系：

$$F(i,=)=f(i,=)$$

$$F(0,>)=f(0,>)$$

$$F(i,>)=F(i-1,>)\ \text{OR}\ f(i,>)$$

$$F(L,<)=b(L,<)$$

$$F(i,<)=F(i+1,<)\ \text{OR}\ f(i,<)$$

即 $F(i,op)$为一个表示所有(i,op)对应约束满足情况下满足的包含 op 操作约束集合的布隆过滤器。

Num	>	<	=	Constrains	Bloomfilter
-45.6	{5647, 7898}	{3611, 5564, 6617, 7118, 7130, 9142}	0	{">", -45.64}	{5647, 7898}
-38.3	{5647, 7898}	{3611, 5564, 6617, 7118, 7130, 9142}	0	{">", -33.90}	{2632, 6297}
-33.9	{2633, 5647, 6297, 7898}	{3611, 5564, 7118, 7130}	0	{">", -26.93}	{3005, 5010}
-26.9	{2632, 3005, 5010, 5647, 6297, 7898}	{3611, 5564, 7118, 7130}	{3540, 8132}	{">", -20.93}	{4189, 9473}
-17.1	{2632, 3005, 5010, 5647, 6297, 7898}	{3611, 5564, 7118, 7130}	0	{">", -13.59}	{2023, 7631}
-13.6	{2023, 2632, 3005, 5010, 5647, 6297, 7631, 7898}	{5564, 7130}	0	{"=", 5.99}	{3912, 8907}
5.99	{2023, 2632, 3005, 5010, 5647, 6297, 7631, 7898}	{5564, 7130}	{3912, 8907}	{"=", 14.50}	{2843, 8407}
14.5	{2023, 2632, 3005, 5010, 5647, 6297, 7631, 7898}	0	{2843, 8407}	{"=", -37.58}	{8133, 9543}
20.93	{2023, 2632, 3005, 4189, 5010, 5647, 6297, 7631, 7898, 9473}	0	0	{"=", -26.93}	{3540, 8132}
37.58	0	0	{8133, 9543}	{"<", 5.99}	{5564, 7130}
				{"<", -38.26}	{6617, 9142}
				{"<", -17.08}	{3611, 7118}

图 3 数字测试样例及数据结构

以图 3 为例，如果要检查值 0.5 满足哪些约束，首先通过二分法找到 0.5 的位置，即 −13.59 和5.99之间。(5.99,<)位置的布隆过滤器表示的集合包含所有"<"操作相关约束中数值 0.5 可以满足的约束，同时(−13.59,>)位置的布隆过滤器表示的集合包含数值 0.5 满足的所有">"操作相关的约束。在对这两个值进行 OR 位操作之后，便可以得到表示数

值0.5满足的所有约束集合的布隆过滤器。如图3中,0.5可以获得布隆过滤器{2023,2632,3005,5010,5564,5647,6297,7130,7631,7898},这个布隆过滤器表示集合{{>,-13.59},{>,-33.90},{>,-26.93},{<,5.99},{>,-45.64}},即数值0.5满足所有约束的集合。

将不同的运算集中在一张表中可以获得更好的时间和空间效率,使单次查找过程的时间复杂度是O(ln(K))。假设对于某个属性,其3种运算对应的约束数量分别为L,M,N,则有:

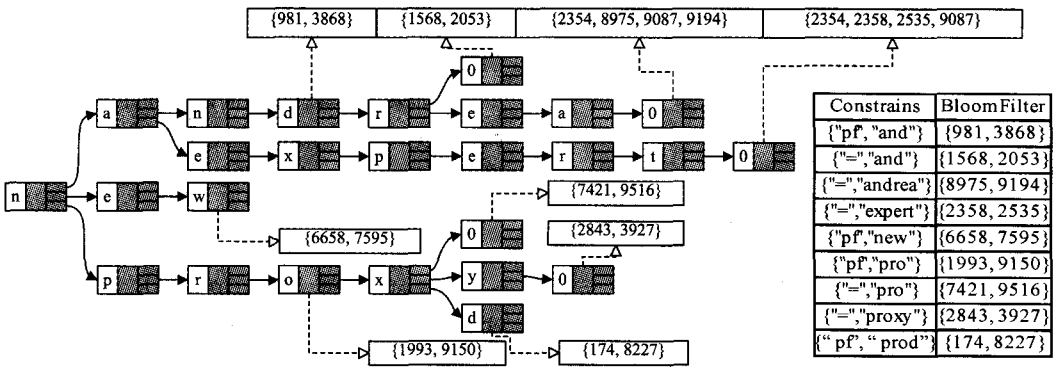
$$L+M+N>=K$$

若分别组织成表,则所需查找次数为$\ln(L)+\ln(M)+\ln(N)=\ln(LMN)$大于$\ln(K)$。另一方面,对拥有相同数值不同操作的约束,也要多存储几次数值。

5.3 字符串类型测试

定义在字符串上的"="、"PF"和"IN" 3种操作也可以同时进行测试。对于每一个属性,算法维护一棵三叉树(Ternary Search Tree,TST)[12]。约束中出现的每个字符串以"\0"结尾存储在三叉树中,包含"\0"字符的节点称为0节点。树中的每个节点包含3部分信息:当前节点表示的字符、1个布隆过滤器以及3棵子树。从树根到某个0节点的路径可以表示一个字符串,到某个非0节点的路径可以表示一个字符串的前缀。若某个节点表示的字符串为s,对非0节点,其布隆过滤器表示约束{PF,s},而对0节点,其布隆过滤器包含约束{=,s}和所有的{IN,ss},其中s在集合ss中。

测试时,算法首先生成一个全0布隆过滤器。在搜索过程中,搜索路径上每一个匹配的节点中的布隆过滤器都被更新到结果中(使用OR操作),如图4(全0布隆过滤器被省略)所示,如果搜索字符串"andrea",将会得到布隆过滤器{981,2354,3868,8975,9087,9194},即集合{{PF,"and"},{IN,{"andrea","expert"}},{=,"andrea"}}。



Constrains	BloomFilter
{"pf","and"}	{981,3868}
{"=","and"}	{1568,2053}
{"=","andrea"}	{8975,9194}
{"=","expert"}	{2358,2535}
{"pf","new"}	{6658,7595}
{"pf","pro"}	{1993,9150}
{"=","pro"}	{7421,9516}
{"=","proxy"}	{2843,3927}
{"pf","prod"}	{174,8227}

图4 字符串测试样例及数据结构

该字符串测试算法的速度是O(H),其中H是TST树的高度。由于TST树相对固定且在建树时就已获得全部信息,因此可以使用二分法等方法创建最优树以提高算法效率。

5.4 订阅匹配

对所有属性进行测试之后将产生一系列的布隆过滤器,对这些布隆过滤器进行集合的并操作(按位OR)可以得到一个新的布隆过滤器。该布隆过滤器表示的集合包含该对象满足的所有断言。一个订阅是其中所有断言的集合。要计算一个对象是否满足一个订阅的全部条件,只需要计算订阅对应的断言集合是否是对象满足所有断言集合的子集。设订阅的布隆过滤器为A,算法生成的布隆过滤器为B,匹配结果为R,则有:

$$R=(\text{NOT}((A\ \text{AND}\ B)\ \text{XOR}\ B))$$

其中,NOT、AND和XOR是按位取非、按位与和异或操作,若结果R为0,则匹配失败,反之则匹配成功。

获得了一个表示对象满足所有断言集合的布隆过滤器之后,匹配功能块将该布隆过滤器和所有订阅的布隆过滤器进行匹配。找出匹配的订阅并推送对象ID给其目标地址。

在匹配订阅时,本文设计不使用Jerzak[13]提出的bftree或sbstree,而是直接进行逐条匹配,因为:

1. 在实际应用场景中,被匹配的订阅数量经常很多,例如某一传感器数据被多方应用。如果将所有订阅组织成bftree或sbstree,需要在树中往复遍历,增加了算法和实现的复杂度。

2. 尽管要进行匹配的订阅数量较多,但计算只是基本的位操作的组合,实际计算速度很快。

值得注意的是,匹配过程中几乎全部运算都是简单的位运算,算法代码过程中不涉及分支。接下来的研究工作中,我们考虑使用GPU甚至设计专用硬件直接进行并行计算。

以GPU为例,对于大量数据的相同运算,GPU可以获得远高于CPU的运算速度。这里的匹配算法便是这种情况。考虑到显存和内存的数据交换是性能瓶颈,可以将相对固定的众多订阅的布隆过滤器存放在显存中,以减少显存和内存之间的数据传送量,每次将新获得的布隆过滤器传入显存,并行地与各个订阅的布隆过滤器进行比较,并输出匹配成功的订阅索引。

结束语 本文提出了一种数据分发网络及相应算法。方案借用大规模存储系统节点网络构建数据分发网络,通过收集存储系统中众节点上有限的计算资源,为应用于物联网场景的存储系统引入了基于内容的数据分发机制,使得各种传感数据能够更有效地聚合和分发。受系统虚拟化技术和云计算的启发,本文引入工作者和功能块的设计,通过工作者和功能块的动态装配组合实现了计算节点的负载自动调节和计算资源在不同任务间的自动配置。最后,本文还给出了基于布隆表达式的一种快速约束测试和匹配算法。

参考文献

[1] Pietzuch P,Muhl G,Fiege L. Distributed Event-Based Systems: An Emerging Community[J]. Distributed Systems Online,IEEE,2007,8(2):1-3

(下转第90页)

文献[4]中所提方法由于只重视链路的拥塞，且没有依据时间与拓扑的动态变化，因此如图6、图7所示，当网络流量较少时，普通优化由于只重视拥塞控制，因此在负载均衡上的性能优于本算法，但当网络流量逐渐增加时，本算法也逐渐重视拥塞，双方负载均衡能力相当，且本算法的传输代价一直小于普通算法。因此本算法在处理多拓扑动态多目标负载均衡上有一定优势。

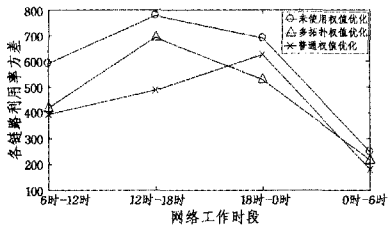


图6 链路利用率方差

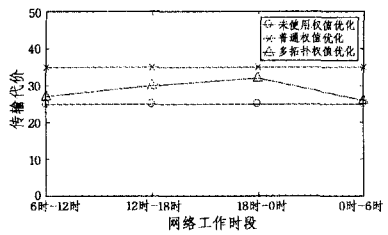


图7 各时段传输代价

结束语 本文根据网络流量动态变化的特点，提出一种MTR子层链路权值优化算法。该算法根据时段不同设置相应的业务量矩阵，且因为网络流量不同，对拥塞和传输代价的偏重程度也不同，因此该算法为不同的时段设置了不同的拥塞、传输代价权重因子，使得网络MTR子层可以在不同时段拥有一套适应该时段业务需求的链路权值。为了避免一般优化算法容易出现陷入局部最优的问题，该算法使用小生境粒子群算法进行寻优，以保证搜索的范围能够覆盖全局。实验证明，该算法能够实现子层中流量的负载均衡。在后续研究中，将对拥塞造成的代价函数进行优化，使其更加适应网络环境。

参考文献

[1] Fortz B, Thorup M. Internet traffic engineering by optimizing OSPF weights[C]// Proceedings of IEEE infocom 2000. Te-Aviv, Israel, 2000: 519-528

[2] Fortz B, Thorup M. Optimizing OSPF/IS-IS weights in a changing world[J]. IEEE Journal on Selected Areas in Communications, 2005, 20(4): 756-767

[3] Karthik L, Matthew C, Murali R, et al. Achieving convergence-free routing using failure- carrying packets[J]. Proceedings of ACMSIGCOMM, 2007, 37(4): 241-252

[4] 于涛，陈山枝，李昕，等. 一种通过优化链路权值来增强网络生存性的方案[J]. 高技术通讯，2008，18(7)：661-665

[5] 吴静，郭成城，晏蒲柳，等. 基于动态流量矩阵的网络链路权值调整方法[J]. 华中科技大学学报，2007，35(7)：76-79

[6] 袁荣坤，孟相如，李明迅，等. 基于粒子群权值优化的网络可生存性增强方法[J]. 计算机应用，2012，32(1)：127-130

[7] 黄赫，王晟. 多拓扑路由实现IP网络区分服务的优化算法[J]. 计算机应用研究，2010，27(12)：4735-4737

[8] Brits R, Engelbrchta P, Bergh F D. A niching particle swarm optimizer[C]// Proceedings Conf. on simulated evolution and learning. Singapore, 2002: 1037-1040

[9] Scheffel M C, Gruber C G, Schwabe T, et al. Optimal multi-topology routing for IP resilience[J]. International journal of electronics and communications, 2006(60): 35-39

[10] 徐明伟，杨芫，李琦. 域内自愈路由研究综述[J]. 电子学报，2009，37(12)：2753-2761

(上接第68页)

[2] Zhang Z, Lian Q, Lin S, et al. BitVault: A highly reliable distributed data retention platform[J]. ACM SIGOPS Operating Systems Review, 2007, 41(2): 27-36

[3] Kubiatowicz J, Bindel D, Chen Y, et al. OceanStore: An Architecture for Global-Scale Persistent Storage[J]. ACM SIGPLAN Notices, ACM, 2000, 35(11): 190-201

[4] Costa P, Picco G. Publish-subscribe on sensor networks: a semi-probabilistic approach[C]// Mobile Adhoc and Sensor System Conference. Washington, DC: Costa P, 2005: 323-332

[5] Carzaniga A, Hall C P. Content-based communication: a research agenda[C]// Proceedings of the 6th international workshop on Software engineering and middleware. 2006: 2-8

[6] Baldoni R, Virgillito A. Distributed Event Routing in Publish/Subscribe Communication Systems: a Survey[J]. Communications Surveys & Tutorials, 2010, 12(1): 39-58

[7] Barham P, Dragovic B, Fraser K, et al. Xen and the art of virtualization[C]// Proceedings of the nineteenth ACM symposium on Operating systems principles. 2003: 164-177

[8] Kivity A, Kamay Y, Laor D, et al. kvm: the Linux virtual machine monitor[C]// Proceedings of the Linux Symposium. 2007: 225-230

[9] Moreno-Vozmediano R, Montero R, Llorente I. IaaS Cloud Architecture: From Virtualized Data Centers to Federated Cloud Infrastructures[J]. Computer, 2012(99): 1

[10] Bhardwaj S, Jain L. Cloud computing: A study of infrastructure as a service (IAAS)[J]. International Journal of engineering and and Information Technology, 2010, 2(1): 60-63

[11] Knuth D. The art of computer programming [M]. Volume 3, 1973

[12] Bentley J L, Sedgewick R. Fast algorithms for sorting and searching strings[C]// Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms. 1997: 360-369

[13] Jerzak Z, Fetzer C. Bloom filter based routing for content-based publish/subscribe [C]// Proceedings of the second international conference on Distributed event-based systems. 2008: 71-81