低成本分布式邮件备份系统的设计和实现

刘 波,杨 强

(趋势科技中国研发中心,江苏 南京 210012)

摘 要:为了利用廉价的普通桌面电脑构建高性能、高可靠、低成本的电子邮件备份系统,设计了基于SOA的对等网络存储架构,贯彻了低耦合、无状态、异步通信和自适应的设计原则。系统将输入数据流和输出数据流分开处理,分别将邮件数据和索引信息保存在磁盘文件和数据库中,兼顾了高容量和快速查找的的要求。

关键词:电子邮件;备份;分布式存储;SOA;低成本

中图分类号:TP311 **文献标识码**:A **文章编号**:1674-7720(2012)05-0079-05

Design and implementation of a low-cost distributed e-mail backup system

Liu Bo, Yang Qiang

(TrendMicro China Development Centre, Nanjing 210012, China)

Abstract: In order to use cheap desktop computers to build high-performance, high reliability, low-cost e-mail backup system, we designed the SOA based peer to peer network storage architecture. It implemented our design principles included loosely coupled, stateless, asynchronous communication, and adaptive. The input stream and output stream are separated in the new architecture. Taking into account high capacity and fast search, the message data and the index data stored in disk files and database separately.

Key words: e-mail; backup; distributed storage; SOA; low-cost

在一个在线电子邮件扫描系统中,需要设计一个安全可靠的存储系统,暂时保存被隔离的电子邮件。这些邮件一直保留在系统内,直到用户选择释放或者到期失效。这是一个需要精心设计的系统,考虑牵涉到的数据量和在线系统所要求的高可靠性,如何用尽可能低的成本实现这个系统,成为项目团队面对的主要挑战。

1 分布式邮件备份系统的技术需求和设计原则

邮件备份系统最基本的要求是高吞吐量,具体来讲,要求能达到每秒1 000封邮件(平均大小10 KB)的最大输入速度,用户的邮件要能保存最多60天,系统总容量在20 TB以上,用户的检索速度应该控制在1 s以内。

作为一个商业运营的在线系统,除了具备很高的性能以外,还要具备很高的可靠性和尽可能低的成本。系统的SLA[1](Service Level Agreement)要达到99.99%以上,也就是说运行一年的停机时间必须控制在52 min以内。在线系统的主要成本包括数据中心的成本和运行维护成本。数据中心的成本主要由硬件成本、网络成本、其他成本如电费、冷却、机架空间等构成,而运营成本主要由运营维护、技术支持等费用构成。依据上述基本要求,确定了邮件备份系统的基本技术需求:(1)单机高效率,系统高吞吐;(2)使用低成本硬件,避免使用昂贵硬件;(3)提供高可伸缩性;(4)部署自动化;(5)管理智能化。

根据上述需求,确立系统整体设计的原则,主要包括下述四点:

(1)低耦合。耦合是指系统的子模块之间结合的紧密程度,一个低耦合的系统不但要降低模块之间的静态依赖关系,而且要尽可能地降低模块在时间和空间上的依赖关系。低耦合的系统可以降低模块之间的相互影响,提高系统的扩展性。

(2)无状态。无状态的服务不在内部保留状态数据,上一个请求和下一个请求之间没有任何关系。无状态的服务更容易实现水平扩展。

(3)异步通信。异步通信可以提高通信的效率,请求发送者无需等待服务的返回,节点之间不需要互相等待,可以提高每个节点的处理效率。

(4)自适应。自适应的系统能够自动调整、优化运行参数、应对运行环境的变化。能够自我调整,也就减少了人为干预的要求,降低了维护费用,而且自动调整显然比人为干预反应更快、也更准确,所以也提高了可靠性。

2 分布式邮件备份系统的设计

2.1 整体框架设计

系统的整体设计采用SOA[2]的架构,系统由三种不同的角色构成,其关系如图1所示。

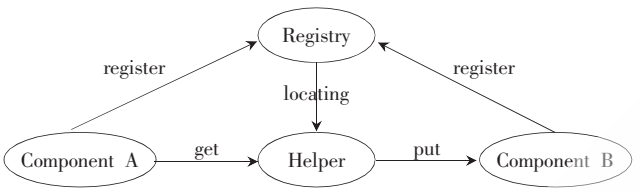

图1 基于SOA的设计

组件A和B完成系统特定的某些功能,A和B对外部提供Restful[3]风格的Web应用接口,但是A和B之间互相没有调用关系,也不知道彼此的存在。Registry是一个注册服务器[4],所有的组件都在Registry注册,并不断通过心跳信号更新自己的状态到Registry;而且Registry维护一张系统所有组件的状态表,对于超时没有更新的组件,就将其从列表中删除。Helper负责将数据从一个组件传送到另一个组件;可以从Registry得到当前系统的所有组件清单以及组件之间的逻辑联系关系,并据此进行数据转运工作;使用标准的HTTP命令从一个组件的输出队列得到数据,然后同样用HTTP命令将数据发送到下游的组件。

同一类型的组件可以同时有多个实例存在,以提供高可靠性和任务负载的均衡扩充。Helper本身也可以有多个实例存在,提供多个上游组件和下游组件之间的动态数据交换和负载均衡。

组件工作在异步的模式下,每个组件都有一个输入队列和一个输出队列,需要处理的数据通过Web接口被放进输入队列,组件的数据处理程序依次从输入队列读取数据,处理完成后放入输出队列,然后通过Web接口被取走。组件拥有完成处理工作所需要的所有资源和数据,可以独立完成自己的任务。这是保证组件异步、高效工作的重要前提条件。

一个最简单的系统至少包括三种不同的功能组件,它们分别是Generator、Processor和Terminator。它们之间的关系如图2所示。

Generator是任务的发起方,一个Generator至少有一个任务输出队列和一个对应的ACK输入队列。Generator产生的任务数据在任务输出队列中排队等候Helper转送到Processor进行处理。一旦一个任务被送到

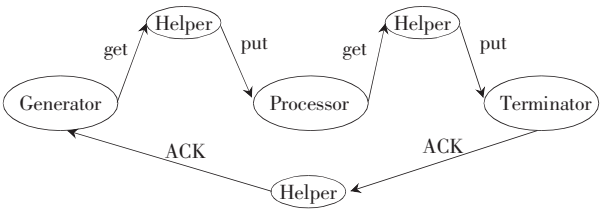

图2 基本处理流程

Processor,这个任务就被放入一个等候确认的队列,等候任务完成的确认。Processor至少有一个任务输入队列和一个结果输出队列,即从任务输入队列取得待处理任务,进行处理后放入结果输出队列,由Helper送到Terminator。Terminator对结果进行最后的处理后,生成对应的ACK,ACK被送回Generator,完成一个完整的任务处理生命周期。如果Generator在规定时间里没有收到一个任务的ACK,说明这个任务可能在系统内被丢失了,按不同的业务逻辑,Generator可以重新发送这个任务,也可以丢弃这个任务。确认-重发机制保证了异步系统中数据的可靠性问题,较好地解决了节点失效时丢失任务的问题。此外,通过Helper桥接的设计允许在一个任务处理序列里组合多个不同的Processor来完成复杂的任务而不用修改程序,这种重组甚至可以在运行中的系统上动态完成,大大提高了系统的弹性。

图3是邮件备份系统的架构图,主要的组件是QtStorage和QtDAL。QtStorage负责存储邮件的原始信息(包括邮件的原文、发送者、接收者等信息),将收到的邮件写入存储区域后,会将邮件及相关信息编制成摘要。QtDAL负责将邮件的摘要存储在数据库中,并为用户提供检索服务。为了实现高效率的运作,存储系统的设计采用了读写分开的思想。大量原始数据的写入在QtStorage上完成,QtStorage拥有多个实例,可以分散写入的压力,同时也能够提供多重冗余,实现高可用性。QtStorage可以实现无缝的横向扩展,其存储能力和容量随之扩展。经过QtStorage处理过的摘要信息,其容量已经大大缩减,所以可以采用单一的索引节点来保存,单一节点可以以较低的代价实现数据的一致性。这里的QtDAL就是这样的一个索引节点,用户的查找和读取主要发生在QtDAL上,查找和读取索引信息和数据的写入发生在不同的节点上,这样就实现了数据的读写分开,提高了运行效率。只有在用户需要取回被隔离的原始邮件时,才会到QtStorage上去读取,按照系统的设计逻辑,这是一个概率极低的操作,不会对系统构成很大的影响。

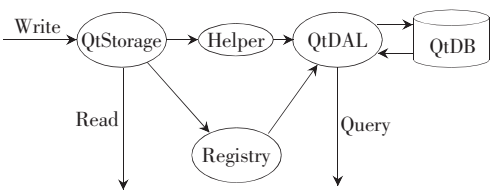

图3 系统架构图

2.2 文件存储服务

文件存储服务为整个系统提供原始数据的存储服务。根据经验,这部分的成本是整个系统成本的主要构成部分。为了保持低成本,必须避免使用价格昂贵的高性能存储设备,如 SAN、NAS 等。而且为了便于扩展和维护,一些需要特别硬件支持的存储设备也不能考虑,如 RAID 阵列等。一个可行的设计思路是使用最廉价的 PC 和随机安装的硬盘来构成系统。商用的 PC 可以安装 4 个处理核心的 CPU 和 2 TB 以上的硬盘,其性能和容量已经足以满足系统的需求。但是不论是 PC 的硬件还是硬盘本身,其可靠性和稳定性都不能达到 99.99%的可靠性。解决的方法是用多重冗余的方式构建系统,不但数据的保存是多重冗余的,而且处理节点也是多重冗余的。

2.2.1 数据的写入

图 4 是文件存储系统的设计。Storage 1~3 是 3 个存储节点,各自带有 2 TB 的硬盘存储数据,并对外开放 3 个 Web 接口,分别是接收文件输入的 Input Queue、输出文件索引信息的 Index Queue 和输出文件同步包的 Sync Queue。一个文件的存储过程可以用以下步骤描述:

(1)Helper 随机挑选一个 Storage 节点(例如 Storage 1),将待保存的文件及控制信息一起发送到 Storage 1 的 Input Queue。

(2)Storage 1 将文件存盘,生成文件的索引信息,将索引放到 Index Queue。

(3)Storage 1 修改文件控制信息,将文件和控制信息添加到 Sync Queue。

(4)Helper 从 Storage 1 的 Sync Queue 中将待同步文件取出,送到另一个 Storage 节点(例如 Storage 2)。

(5)Helper 将索引信息从 Storage 1 取出,送到数据库保存。

(6)Storage 2 重复上述步骤,直到重复存储次数达到需要的冗余度为止。

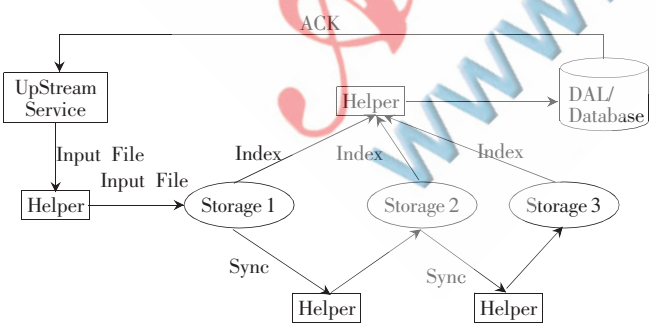

图 4 文件存储流程

这个设计的主要特点如下:

(1)每个 Storage 节点都同时具有处理能力和存储容量,添加节点就可以同时扩展处理能力和容量,系统的扩展是对称的,非常利于管理和维护。

(2)Storage 节点之间没有相互依赖性,任何一个 Storage 节点因为故障下线,都不会对整个系统产生影响。

(3)每个文件需要多少冗余度、是否加密、是否压缩,都可以单独控制,非常灵活。

(4)添加新的 Storage 节点后,负载会自动均衡到新的节点上,有利于资源的有效利用。

(5)系统要求的保存文件期限最多 60 天,所以在 3 冗余或更多冗余的情况下,可以不用考虑损坏节点的文件恢复功能。即使使用可靠性较低的普通 PC 和 SATA 硬盘,也可以保证文件在 60 天内的可用性。

(6)读写分开,大负荷的文件写入被分散到数量众多的 Storage 节点上,数据库节点只负责少量、偶尔的用户查询操作,提高了硬件效率,降低了成本。

(7)系统并不是严格一致的,通过牺牲部分一致性,获得了较好的扩展性和可用性,是 CAP[5]原则的具体实践。

(8)Storage 节点之间是没有联系、没有交互的,所有的数据交换都通过 Helper 进行。文件的控制信息随同文件数据本身传送,没有任何一个节点管理和追踪文件的去向,各个节点间通过合作完成任务,没有中心控制节点。这个过程充分体现了“低耦合”、“异步”、“无状态”的设计原则。

需要说明的是,Storage 节点的数目建议大于系统设置的最高冗余度。例如对于 3 冗余的系统,Storage 节点建议至少设置 4 个,这样,在某一个节点因故下线的时候,系统仍然可以正常运行。当然,设置更多的节点可以获得更好的可靠性和性能。

2.2.2 数据的查询

数据的查询相当简单,按照用户提供的查询条件,可以在数据库中查询到符合条件的记录,每条记录中都包含对应的数据文件在 Storage 节点上的位置。如果用户需要读取原始数据,可以调用 Storage 节点上的 API 接口,提供位置信息,读取原始数据。记录中包含文件的多个冗余备份的位置,如果一个节点失效或者不能访问,可以读取备用节点上的信息。

2.2.3 数据的删除

数据的删除分为数据库记录的删除和数据文件的删除,数据库节点和 Storage 节点都有内置的定时任务,可以按时间删除已经过期的数据。

2.2.4 系统容量的扩展和故障节点的替换

当文件存储系统的容量不能满足要求时,需要对系统容量进行扩展,这种扩展应该是一种简单的、无缝的操作,不能干扰系统的正常运行,不需要修改软件,不需要停机以及尽量少的配置改动。本系统较好地实现了上述要求。

2.3 索引数据库

索引数据库保存着所有备份邮件的索引信息和位

置信息。这是一个标准的关系型数据库，采用开源的PostgresSQL实现。数据库主要包括两张表，如图5所示。

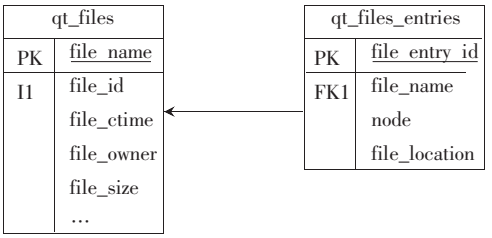

图5 数据库表

主表qt_files保存邮件的摘要信息，例如邮件主题、发送时间、发送者等，用户可以提供条件快速查询到符合的邮件。qt_file_entries表里存放邮件具体的备份位置，包括所属节点、在节点里的定位信息等。每封邮件对应有多个位置记录，都通过外键联接到主表上。

3 性能测试结果及分析

3.1 测试系统配置

为了尽量降低成本，本文选用普通的PC代替昂贵的服务器完成处理工作。在这个测试中，采用了4台DELL 780MT构成系统，CPU为Intel Q8400，主频2.66 GHz，内存8 GB，配备2 TB的SATA硬盘。4个节点通过千兆交换机连接。

3.2 性能数据

测试中，通过前端不断地灌入邮件，记录系统能够稳定持续处理的最高上限，即为系统的最大性能点。为了考察不同大小的邮件对性能的影响，每种配置下，都用10 KB和30 KB的邮件测试两组数据。

图6是单节点配置下的性能数据，在单节点情况下，10 KB的邮件速率达到1 800封/s，30 KB的则达到1 400封/s，完全达到了系统的设计要求。

图6 单节点性能数据

为了提高数据的安全性，可以设置将备份的邮件在不同节点上拷贝多份冗余。图7即为冗余度设置为2时的测试数据，考察了不同节点数量对性能的影响。可以看到，随着系统节点数的增加，系统性能也呈基本线性地增加，这充分反映了节点间异步通信的重要性。异步通信削弱了节点间在时间上的依赖关系，通过牺牲局部的、临时的一致性，充分发挥了每个节点的处理能力，提高了系统的效率。

为了进一步提高数据可靠性，可以把冗余度设置到3，

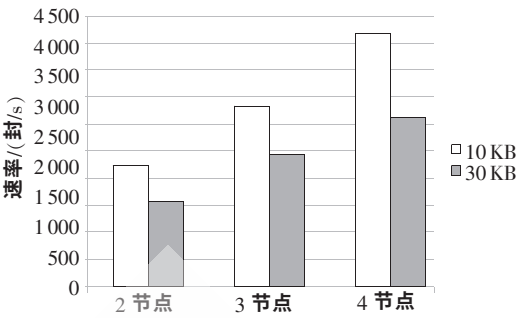

图7 2冗余性能数据

此时每封邮件在系统内有三份拷贝，达到了极高的可靠性。如图8所示，在4节点3冗余的测试环境下，系统的吞吐速率达到了3 000封/s（10 KB邮件）和2 000封/s（30 KB邮件）。如果等价成单个节点的平均速率的话，分别达到2 250封/s和1 500封/s，比单节点的性能还要好些。这是因为随节点的增多，不但存储空间得到扩展、可靠性得到增加，CPU的数量和处理能力也得到加强，更多的CPU的加入，也在一定程度上提高了系统的整体处理能力。

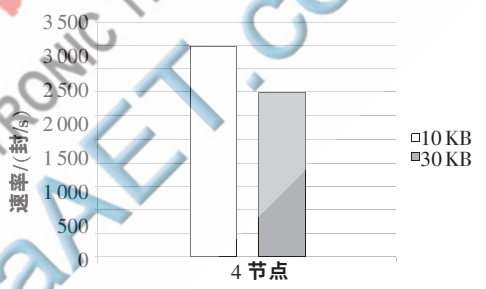

图8 3冗余性能数据

3.3 结果分析

通过对结果的分析，发现在单节点场景下，采用30 KB邮件样本时系统达到了42 MB/s的写入速度，而10 KB邮件的写入速度为18 MB/s，两种测试环境中CPU占用均为30%左右，说明系统瓶颈不在CPU和I/O中。通过对代码实现的审查发现，问题主要是同步写文件导致的，对文件的同步读写操作降低了I/O的效率，并且随样本的缩小，读写操作的频繁化而更趋恶化。这就是为什么小样本性能不成比例增加的根本原因。全面采用异步磁盘I/O操作可以很好地改善这个问题，但是异步I/O程序远较同步I/O复杂，而且在本项目里，即使是同步I/O也足够满足性能需求，所以本设计最后决定保持目前的代码不变。

比较不同组合条件下的测试结果可以发现，性能随冗余度的增加而降低、随节点数目的增加而提高，三者间的关系接近线性关系。如果节点数目增加一倍，性能也能够增加一倍。增加冗余度会降低性能，但可以通过增加节点来补偿。同时添加节点数量和冗余度，能够得到更好的可靠性和性能，并且收益并不随节点数目的增加而递减。对于一个分布式系统，这是一个非常可贵的

特质。因为管理者可以通过简单的添加处理节点，获取更多的存储和处理能力。

为了最大限度地降低成本、提高可靠性，在设计的初期阶段就确立了低耦合、无状态、异步通信和自适应这四个设计准则。通过放宽对一致性的要求，仅保证最终一致性，充分解耦各个节点间的依赖关系，获得了优异的可靠性和可扩展性。最终的测试结果也证明了这种设计思路的正确性，采用普通桌面计算机和廉价磁盘组成的系统足以在容量、性能、可靠性和可扩展性方面满足要求，而成本仅为传统方案的十分之一。

参考文献

[1] TANENBAUM A S, STEEN M V. Distributed systems principles and paradigms[M].杨剑锋，常晓波，李敏，译，北京：清华大学出版社，2004.

[2] JOSUTTIS N M. SOA in practice[M].程桦，译.北京：电子工业出版社，2008.

[3] ROCHARDSON L, RUBY S. RESTful Web Services 中文版[M].徐涵，李红军，胡伟，译.北京：电子工业出版社，2008.

[4] 杜宗霞，怀进鹏，主动分布式 Web 服务注册机制研究与实现[J]. 软件学报，2006，17(3)：454-462

[5] BREWER E A. Towards robust distributed systems. (Invited Talk)Principles of Distributed Computing, Portland, Oregon, July 2000.

(收稿日期：2011-11-09)

作者简介：

刘波，男，1973 年生，硕士，主要研究方向：云安全系统的设计与实施。

杨强，男，1983 年生，本科，主要研究方向：Saas 系统的性能调优，性能测试。