

负载均衡技术的研究与实现

喻 莉,阮文涛

(华中科技大学 电子与信息工程系,湖北 武汉 430074)

摘 要:为了解决系统服务能力的巨大变化及节省系统升级的费用,采用并详细分析了服务分担模式的工作原理,并在此原理的基础上采用虚 IP 技术以及最小公倍数策略或倍数策略,尽可能大地提高系统的处理能力以及高稳定性,以此来使系统的性价比尽可能地达到最大。通过实验表明,文中采用的策略非常适合于中小型服务量的系统。

关键词:服务分担模式;N+K 冗余模式;HA 模式;虚 IP;高可用性

中图分类号:TP311

文献标识码:A

文章编号:1673-629X(2007)08-0120-03

Research and Implementation of Load Balance Technology

YU Li, RUAN Wen-tao

(Department of Electronics and Information, Huazhong University of Science and Technology, Wuhan 430074, China)

Abstract: To realize the goals of easy to change the system's serving capability and low expense to upgrade system, analyzes the principle of load-sharing work mode in detail, and adopts virtual IP technology and lease common multiple or multiple policy with load-sharing work mode to maximize the serving capability and availability of whole system, but minimize the cost of whole system. The experiment result shows this strategy is pretty suitable for medium size or less of service.

Key words: load-sharing mode; N+K redundancy mode; HA mode; virtual IP; high availability

0 引 言

当前,无论在企业网、园区网还是在广域网如 Internet 上,服务器必须具备提供大量并发访问服务的能力。如果将服务器组成一个系统,并通过软件技术将所有请求平均分配给所有服务器,那么这个系统在不考虑网络流量的情况下就完全拥有每秒处理几百万个甚至更多请求的能力^[1]。如果以后的服务量减少了则只需要减少几个服务器和进行很小的配置改动就可以将闲置的处理能力用作它途。这样服务运营商可以很方便地依据业务量的大小调整服务器的数量。把这种弹性的工作模式称为服务分担模式或 N+K 冗余模式。它和下文要谈的 HA 模式一起被称为冗余模式。

1 调度策略分散方案

当前的服务分担方案都要使得调度中心去检测或变相地检测集群服务器能否正常工作以及负载情况,这样势必增加调度中心的负担。如果服务器正常工作与否由集群自己去解决且该工作增加集群服务器很

少的负担,这样就能够减轻调度中心的负担。为此,在应用中设计了调度策略分散方案。

调度策略分散方案的主要思想是:为了减少调度中心的负担,调度中心并不监控集群服务器工作情况,而只需知道有哪些 IP 地址提供服务,也就是提供服务的集群服务器 IP 地址^[2]。如果这些 IP 地址是实 IP 地址的话,那么一旦有服务器不能正常工作,脱离了集群,调度中心还会继续给这个服务器下达任务,这样就影响用户的服务质量。怎样解决这个问题呢?采用动态绑定虚 IP,即使所有集群服务器绑定虚 IP,一旦集群中有服务器不能正常工作,程序就会自动地将虚 IP 释放掉,其它能正常工作的服务器就会把被释放掉的虚 IP 绑定起来。这样,这些虚 IP 在任何情况下都是被绑定在能正常工作的集群服务器上,而调度中心只需要知道这些虚 IP 就够了。

1.1 调度中心的 IP 地址映射表

要使用调度策略分散方案,调度中心必须有一张含有服务器虚 IP 地址的映射表。为了提高速度,该映射表存放于调度中心的 cache 中。该映射表至少有这么两个字段:一个字段的值存放的是集群服务器所绑定的虚 IP 地址,另一个字段存放对外提供服务的端口号。根据该映射表调度中心就知道了要将客户端消息

收稿日期:2006-11-21

作者简介:喻 莉(1970-),女,江西萍乡人,教授,研究方向为计算机网络、多媒体信息处理、无线通信。

发给相应的服务器进行处理。其结构如表 1 所示。

表 1 IP 地址映射表结构

IP	Port	Option field 1	Option field n
IP 0	Port 1
.....
IP M	Port M

当用户的请求达到调度中心时,调度中心采用轮询或随机的方式从映射表中取一个虚 IP 地址,然后采用 IP 管道技术将虚 IP 封装在用户的消息的 IP 报头上发送出去。

1.2 虚 IP 个数的确定

为了保证所有的服务器都能对外提供服务,我们使用的虚 IP 不得少于服务器的个数,同时为了保证无论有几台服务器不能工作都能够使剩下能正常工作的服务器负担均衡,即分到的虚 IP 的个数一样,采用了最小公倍数策略,即:数 1,2,⋯, n 的最小公倍数(其中 n 为集群中真实服务器的个数)。当然如果服务器个数比较多,虚 IP 也就会很多,例如,如果服务器的个数为 6 个,则虚 IP 为 60 个。所以当服务器个数超过 6 个则建议不要采用最小公倍数策略,可以采用第 2 节中讲到的 N+K 冗余工作模式。如图 1 所示。

要使调度中心的映射表数据不用编辑,必须保证表中的所有虚 IP 地址时刻都被绑定到能正常工作的服务器上。为此必须在这些服务器之间采用组播的方式来相互通信,以获得各个服务器能否正常工作的信息,如果有必要的话还可以使服务器之间同步必要的数据。如果有一台服务器不能正常工作,则其它的服务器将会将其绑定的虚 IP 平分再绑定到自己的端口上。如果有服务器又恢复了正常工作,则其它的服务器将会释放一定数的虚 IP 给恢复了正常动作的服务器,让其绑定。这样虚 IP 的数量以及值始终不变。关于如何绑定虚 IP 设计以下的算法(为了方便,使用的虚 IP 都是连续的):

- (1) 从组播消息获取已经正常工作服务器的总共节点数 M 和各个服务器的节点号(事先由手工配置好的);从本服务器的配置文件获取本服务器的节点号 N,起始虚 IP(简称 SIP)以及总虚 IP 的个数 T。
- (2) 根据步骤 1 所得到的数据计算出本服务器节点号 N 在正常工作服务器节点号的排序号 N'。
- (3) 绑定这 $T/(M+1)$ 个虚 IP: $SIP + T(N' - 1)/(M + 1), SIP + T(N' - 1)/(M + 1) + 1, SIP + T(N' - 1)/(M + 1) + 2, \cdots, SIP + T(N' - 1)/(M + 1) + T/(M + 1) - 1$ 等共 $T/(M + 1)$ 个虚 IP 地址。同时广播一个 ARP 消息,通知被绑定的这 $T/(M + 1)$ 个虚 IP 所对应的物理地址。例如,当总的虚 IP 个数为 6,

有三个服务器,当前的节点号为 2 时,起始虚 IP 是 10.56.4.100,则绑定的两个虚 IP 是 10.56.4.102 与 10.56.4.103.转到步骤 4。

(4) 根据组播的信息,来判断有没有服务器不能正常工作或原先不能正常工作的服务器又恢复了工作,如果有则释放自己所有所绑定的虚 IP 并转到步骤 1。

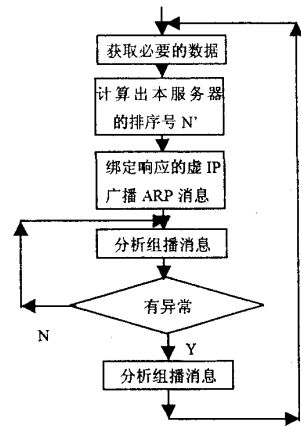


图 1 虚 IP 的绑定与释放流程

由于 IPv4 中 IP 地址空间的日益紧张和安全方面的原因,很多网络使用保留 IP 地址(10.0.0.0/255.0.0.0 等)。这些地址不在 Internet 上使用,而是专门为内部网络预留的。当内部网络中的主机要访问 Internet 或被 Internet 访问时,就需要采用网络地址转换(Network Address Translation, NAT^[3])将内部地址转化为 Internet 上可用的外部地址。NAT 的工作原理是报头(目标地址、源地址和端口等)被正确改写后,客户相信它们连接一个 IP 地址,而不同 IP 地址的服务器组也认为它们是与客户直接相连的。由此,可以用 NAT 方法将不同 IP 地址的并行网络服务变成在一个 IP 地址上的一个虚拟服务。同时装有调度程序与服务程序的服务器要装有两个网卡,一个与 Internet 相连接,一个连接到交换机或集线器里^[4]。

2 HA 模式和 N+K 冗余模式

所谓 HA 模式就是主备模式,即两个功能相同的服务器一个处于活动状态,另一个处于备用状态。这两个服务器都会通过对方的物理 IP 周期地与对方进行通信(我们称作心跳)^[5],目的是检测对方是否正常工作以及更新数据。而这两个服务器使用一个虚 IP 与外界通信,这个虚 IP 被处于活动状态的服务器所绑定,并对外提供服务。如果处于备用状态的服务器检测到对方程序工作不正常(得不到对方的心跳),就会将虚 IP 绑定到自己的端口上,同时广播一个 ARP 消息^[6],通知被绑定的虚 IP 所对应的新的物理 IP。这时

自己就会由以前的备用状态转变成成为活动状态同时对外提供服务。而开始处于活动状态的服务器程序由于工作不正常,就会自动将虚 IP 给释放掉,脱离了整个系统。工作于 HA 模式并不能增加处理用户请求的能力(反而它会增加资金的投入),但是它可以大大提高系统的稳定性,是一种不可忽视的工作模式。

前面提到,如果服务器的个数多于 6 台则可采用 $N+K$ 冗余模式,也就是有 N 台服务器向外提供服务, K 台服务器处于备用状态, N 台服务器绑定 N 的整数倍个虚 IP,一旦处于向外提供服务的服务器不能正常工作,则处于备用状态的服务器应该立即绑定不能正常工作的那个服务器的虚 IP,进入向外提供服务的状态。考虑到经济的原因, K 一般取较小的数字,例如 1, 2 等。系统维护人员会及时地对不能正常工作的服务器进行维修。其它的工作原理同服务分担模式一样。由于会有 1 或 2 台服务器不提供服务,基于经济的考虑,这种模式只有在集群的服务器的个数比较大时才采用。

通过上文的论述,整个系统的工作方式如图 2 所示。

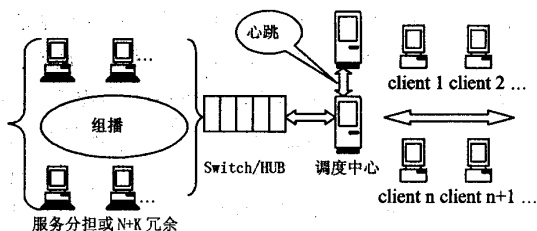


图 2 系统的工作原理图

3 应用实例

在流媒体系统中,为了解决多媒体的数字版权问题,必须对媒体数据进行加密。合法用户在播放媒体时需要到解密密钥分发服务器中取解密密钥。如果在线用户很多,单个服务器就不能满足用户的请求,这时使密钥分发服务器工作于多个服务器的服务分担模式,就可以大大提高处理能力。图 3 是单个服务器与使用调度策略分散方案的负载均衡的两种工作模式的测试比较(服务器是 Linux 操作系统,横轴是每个服务器每秒钟处理的请求个数,纵轴是每条消息的平均响应时间,单位是毫秒)。

说明:测试是在所有的服务器的 CPU 占用率在 75%~85%之间进行的。非正常工作是指服务器(集群)不能及时地响应用户的密钥请求消息。

从图 3 中的数据可以看出:

1)在调度中心模块处理能力范围内,处理用户请

求的总能力与服务器的个数成正比;

2)在调度中心模块处理能力范围内,消息的响应时间几乎没有什么变化。但在服务器的处理极限时(3400Messages/Second·Server)处于服务分担模式的性能会迅速下降。这可能是由于要处理组播消息的缘故。

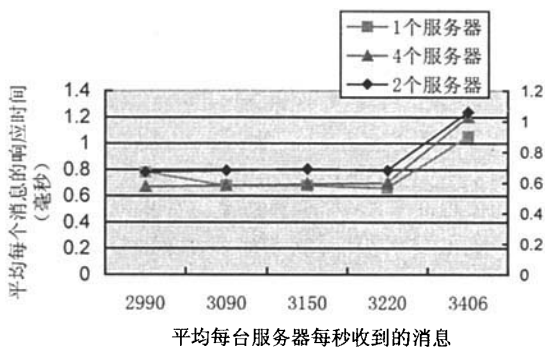


图 3 消息响应时间

从表 2 中的数据可以看出:

(1)一个服务器与两个服务器的时间差不多长。这是因为一旦一个服务器不能正常工作后,所有的请求服务消息都会加在另外一个服务器上,这样这个服务器就得要处理两倍于正常工作的任务,肯定会立刻宕机;

(2)服务器的个数越多,正常工作的持续时间越长,密钥分发系统的稳定性越好。

表 2 处于两种工作模式下服务器(集群)的最大正常工作时间

编号	服务器的个数	正常工作持续的时间	虚 IP 个数
1	单个服务器	15 天	无
2	2 个服务分担	12 天 1 小时	2
3	3 个服务分担	26 天 16 小时	6
4	4 个服务分担	一个月以上	12

其实,一旦工作于服务分担模式下的服务器不能正常工作,管理委员会很快地检查到并进行检修,检修的同时只要服务量不是很大就不会立刻中断,检修好后又可以向外提供服务。

4 小 结

服务分担模式或 $N+K$ 冗余模式在扩充系统的服务能力时无需中断服务,并且服务提供商可以依据业务量的大小调整服务器的数量,提高了系统灵活性,特别是在服务器不是很多的情况下能显著降低开支。此外,操作简单,无需进行人员培训,稳定性也比较强,非常适合中小型服务量(建议是 100 个集群服务器以

(下转第 129 页)

```

<rdf:range rdf:resource="# 科研成果"/>
</rdf:Property>
<owl:Restriction>
<owl:minCardinality rdf:datatype="xsd:NonNegativeInteger">
1</owl:minCardinality>
</owl:Restriction>
...
</owl:Ontology>

```

从上面例子可以看出在 OWL 中引用了 XML Schema 中定义的数据类型,这是因为已知的数据类已经够丰富,没有必要利用本体语言定义新的数据类型,同时 OWL 还继承了 RDF 中的 Rdfs: subclass, Rdfs: Property, Rdf: subProperty, Rdfs: domain, Rdfs: Range 等一系列原语,并且进行了扩展如例子中 owl:Class, owl: minCardinality 等,从而实现了各词汇之间的精确定义。

4 结 论

分析比较了几种语义描述语言,并举例说明了 OWL 是如何描述信息并体现语义的。通过比较可以得出以下结论:XML 可以交换语法,但并没有语义;RDF 是元数据描述框架,在要求有一定语义和推理能

力的情况下使用;OWL 是一种表达能力很强的本体语言,结合了 XML 和 RDF 的优点,可以用于语义要求比较精确的情况。

参考文献:

- [1] Berners-Lee T, Handler J, Lassila O. The Semantic Web[J]. The Scientific American, 2001, 284(5): 34-43.
- [2] Jena B M. A Semantic Web Toolkit[J]. IEEE Internet Computing, 2002(11, 12): 55-59.
- [3] Berners-Lee T. XML and the Web[EB/OL]. 2000-09-06. <http://www.w3.org/2000/Talks/0906-xmlweb-tilb/slide9-6.html>.
- [4] Brickley D, Guha R V. RDF vocabulary description language 1.0: RDF Schema[EB/OL]. 2002. <http://Web.w3.org/TR/rdf-schema/>.
- [5] McGuinness D L, Harmelen F V. OWL Web ontology language overview[EB/OL]. 2004. <http://Web.w3.org/TR/owl-features>.
- [6] W3C. Resource Description Framework Model and Syntax Specification[EB/OL]. 1999. <http://www.w3.org/TR/REC-rdf-syntax/>.
- [7] Gil Y, Ratnakar V. Markup Languages: Comparison and Examples[EB/OL]. 2001-09-07. <http://trellis.semanticweb.org/expect/web/semanticweb/comparison.html>.

(上接第 122 页)

内)。但也存在缺点。例如,如果某个用户的几个请求消息有关联,一旦某个服务器在处理完前一个或几个消息后就不能正常工作,这样用户的后几个关联消息就会发送到其它的服务器上,这样消息处理就会出错。此外方案也只是采用简单的均衡,不能实时地根据各个服务器的实际负载情况加以调整。

参考文献:

- [1] 陈涛,陈启买.分布式计算机系统负载均衡研究[J].计算机技术与发展,2006,16(9):33-34.

(上接第 125 页)

越来越多的人开始使用 Ajax 技术来开发 Web 应用。在应用 Ajax 开发上,Google 公司当仁不让,Gmail, GoogleGroups, Google Maps, Google Suggest 都应用了这项技术,Amazon 的 A9.com 搜索引擎也使用了类似的技术,微软公司也积极地推出了专门的 Ajax 工具——Atlas。通过与众多主流 Web 开发语言的充分结合,又得到了各大 IT 企业的大力支持,Ajax 技术将有光明的前景。

- [2] 肖辽亮. NAT-PT 簇负载均衡的设计与实现[J]. 计算机技术与发展, 2006, 16(3): 80-81.
- [3] 张洪武. 服务器集群与均衡技术研究[D]. 重庆: 重庆大学, 2004.
- [4] 修长虹. 基于 Linux PC 集群负载均衡的研究与实现[D]. 长春: 吉林大学, 2005.
- [5] 尹康凯, 王明伟, 李善平. 高可用性集群中多个节点的心跳模型研究[J]. 计算机工程, 2005, 31(15): 102-103.
- [6] 胡宁, 刘亚萍. 高性能路由器中 ARP 协议关键问题的研究[J]. 计算机工程与科学, 2006, 28(10): 29-30.

参考文献:

- [1] 方俊. Ajax 引擎的设计和应用[J]. 电脑与信息技术, 2006(3): 25-29.
- [2] 苑琛, 曹耀钦. 基于 PHP 技术的网络办公自动化系统[J]. 微机发展, 2003, 13(8): 61-63.
- [3] Flyinghail. Sajax 实例分析[EB/OL]. 2006-10-22. <http://www.flyinghail.net/?p=6>.
- [4] Gehtland J. Ajax 修炼之道[M]. 徐锋译. 北京: 电子工业出版社, 2006.
- [5] 方舟. PHP 技术发展迅猛[EB/OL]. 2006. <http://soft.yesky.com/info/154/2556154.shtml>.