中国科学 : 信息科学　2013 年　第 43 卷　第 1 期 : 60–72
www.scichina.com　info.scichina.com
《中国科学》杂志社
SCIENCE CHINA PRESS
互联网计算的软件技术专刊 · 论文

一种动态的客户端负载均衡机制

王梓又[①②], 周明辉[①②*], 梅宏[①②]

① 北京大学信息科学技术学院软件研究所, 北京 100871
② 高可信软件技术教育部重点实验室 (北京大学), 北京 100871
* 通信作者. E-mail: zhmh@pku.edu.cn

收稿日期: 2012–10–26; 接受日期: 2012–11–20
国家重点基础研究发展计划 (批准号: 2009CB320703)、国家自然科学基金 (批准号: 9118004, 61073016) 和国家高技术研究发展计划 (批准号: 2011AA01A202) 资助项目

摘要　互联网应用的并发用户数量不仅多变并且这种变化常常不可预测. 将系统容量配置为固定值的惯用做法在面临多变的请求时常常会导致两种结果, 一种是因为配置过低而引起用户的不满, 另一种则因为配置过高而造成计算资源的浪费. 而通过运用云架构按需提供、按使用收费的能力, 系统具有了实时地按需配置计算资源的能力. 然而静态的客户端负载均衡方法作为一种主要的负载均衡技术很难适应云架构条件下更加易变的集群结构. 本文提出了一种动态的客户端负载均衡机制. 通过引入分布式的集群视图更新、控制流等技术, 该机制在保持客户端负载均衡机制分布式、可伸缩性强的基础上, 又为集群节点的动态加入与退出和负载均衡策略的动态调整提供了有效的支持. 同时, 本文分析了该机制在一个开源 JEE 应用服务器 PKUAS 中的关键实现问题, 并通过实验从多方面验证了该机制的有效性.

关键词　云架构　动态集群　客户端负载均衡　集群视图　客户端代理

1　引言

随着 Internet 技术的不断发展, 企业级网络应用正在面临着如不可预测的并发用户增长、海量数据管理、系统响应及容量限制等因素的挑战. 网络应用的可伸缩性 (scalability) 与可用性 (availability) 正在受到越来越多的关注 [1,2]. 诸如亚马逊 EC2[1)]/S3[2)] 服务一类的云架构为经济合理地利用计算资源提供了有效的支持. 它们能够按需提供系统容量 (例如服务器数量、存储空间和网络带宽). 应用能够在需要的时候弹性地从提供者处调配计算资源, 而不需要事先为不确定的需求超额配置. 这一特点使得云架构成为了独具特点的架构解决方案, 尤其是在具有可变负载的网络应用中广受青睐. 由于互联网应用的峰值负载和普通负载之间可能存在着巨大的差异, 传统架构很难适应它们的需求. 在传统架构下, 我们要么为潜在的峰值超额配置资源, 从而造成计算资源的大量浪费; 要么只按照普通负载配置, 从而无法应对峰值状况. 运用云架构灵活的配置能力, 互联网应用能够实现实时监控负载情况并且及时获得相应的计算资源.

1) http://aws.amazon.com/ec2/
2) http://aws.amazon.com/s3/

引用格式: 王梓又, 周明辉, 梅宏. 一种动态的客户端负载均衡机制. 中国科学: 信息科学, 2013, 43: 60–72, doi: 10.1360/112012-462

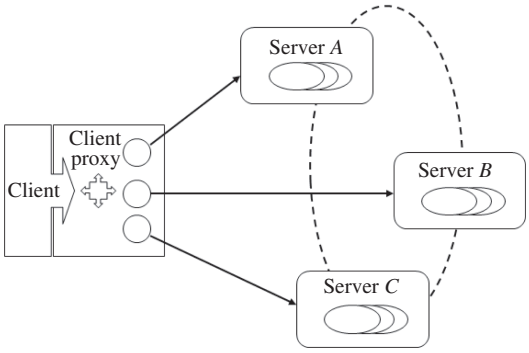

图 1 基于客户代理的客户端负载均衡机制

Figure 1 The client-side load balancing mechanism via the client proxy

目前, 很多成熟的集群技术 [3~5] 都采用客户代理 (client proxy) 的模式作为其负载均衡机制的基础 (图 1). 客户代理维护了一组服务器的引用, 它们分别对应部署了相应服务的节点. 每次客户请求调用时, 客户代理根据选定的负载均衡算法及其他信息 (如各节点的负载加权值) 选择一个节点, 然后向位于该节点上相应的服务发出请求. 这种负载均衡机制的优点主要有以下几点: 首先, 这种方法具有良好的可伸缩性, 负载均衡开销主要由客户负担, 服务器无需考虑由客户数量增加而引起的负载均衡开销的增加; 其次这种方法使用了分布式的负载均衡布局, 避免了集中式负载均衡方法有可能产生的性能瓶颈和单点失效问题.

然而, 客户代理所需要的相关信息往往是在客户端首次连接服务器时通过命名查找、远程代码获取等方法在客户端一次性加载, 而在加载之后, 客户代理只能根据这些静态信息对之后的多次请求进行负载转发. 因此, 在没有发生调用失败的情况下, 无论客户代理存在多长时间, 它所维护的集群视图信息都是不再变化的. 本文将这类方法称为静态的客户端负载均衡机制.

静态的客户端负载均衡机制及其服务器端支撑技术的实现相对简单, 在网络环境稳定及应用请求数量变化不大的情况下效率比较高, 因此能够适应部分早期集群技术的需要. 但是这种机制不能够很好地支持集群视图和负载均衡策略配置的运行时刻调整, 尤其近年来随着集群应用的动态放置 [6,7]、可扩展应用 [8,9] 等云架构技术和集群资源动态分配技术的迅速发展和广泛应用, 它越来越难适应更加广泛的要求. 这主要表现在以下两方面:

首先, 静态的客户端负载均衡机制及其服务器端支撑技术无法良好地支持集群在运行时刻动态的加入和退出节点, 这就意味着很难在服务不间断运行的基础上实现集群资源的动态调整. 例如, 当一个集群负载过大时, 引入新的节点无法被客户端负载均衡机制及时感知到, 从而影响到系统的整体负载能力提升, 并最终导致整个系统吞吐率的下降和响应时间的提高. 另一方面, 当请求量远小于一个集群的负载能力时, 由于无法有效干涉客户端负载均衡机制对服务器的选择, 该集群也很难将闲置的计算资源临时释放出来供其他系统使用, 从而造成计算资源的浪费.

其次, 静态的客户端负载均衡机制及其服务器端支撑技术很难支持负载均衡策略在运行时刻的重配置. 当集群状态发生较大的变化时, 例如部分节点负载能力上升或者下降时, 由于负载均衡机制分布在众多的客户端, 系统无法通过修改负载均衡参数或替换负载均衡策略等方法使各个集群节点的负载量重新达到相对平衡的状态. 例如, 如果集群中某个节点的负载能力由于请求拥塞等原因暂时下降, 而分布在客户端的众多负载均衡器却无从感知这一变化, 所以很容易导致整个系统负载失衡.

针对目前静态的客户端负载均衡机制及其服务器端支撑技术所面临的种种问题, 本文提出了一种

动态的客户端负载均衡机制. 该机制使用了集群技术常用的客户端负载均衡方法, 在保持其良好的可伸缩性的同时, 又通过引入分布式的集群视图更新、控制流等技术解决了它对动态性支持不足的问题. 因此, 该机制可以有效地支持集群节点的动态加入和退出, 并能够良好地支持负载均衡策略的运行时刻重配置. 我们在一个开源的 JEE (Java platform, enterprise edition) 应用服务器 PKUAS 中实现了该机制, 并通过一系列实验从多个方面证明了该机制的有效性.

本文组织如下: 第 2 节对机制的体系结构进行了介绍; 第 3 节描述了在支持客户端动态负载均衡机制时所采用的一些关键流程; 第 4 节分析了该机制在开源 JEE 应用服务器 PKUAS 中的实现; 第 5 节采用实验数据分析了该机制具体实现的有效性; 第 6 节介绍了相关工作; 第 7 节总结全文并探讨了未来工作.

2 体系结构

为了支持动态的客户端负载均衡机制, 本文使用了分布式的集群视图更新方式, 即每个集群节点都能通过监听资源变化及时更新本节点中的集群视图信息, 而客户端则可以选择任意一个集群节点进行集群视图的全部更新操作. 分布式的集群视图更新方式可以保证集群视图的维护与更新不会产生单点失效. 由于客户端集群视图的更新也会产生一定的网络负载, 所以本文使得客户代理可以从任意一个集群节点上获得全部更新, 从而避免了单点失效和网络拥塞等问题. 同时, 本文提出的方法也可以应用于有状态会话和无状态会话两种不同的场景中. 对于无状态会话而言, 每一次新的请求都会被服务器当作是一个独立事件, 所以负载均衡器可以将无状态会话的每一次请求都发送到不同的服务器节点上进行处理. 而对于有状态会话而言, 每一次请求都是带有状态数据的, 如果将一个有状态会话从一个服务器节点上迁移至别的节点处理, 则必须同时将状态数据进行迁移, 这个过程通常会消耗大量的计算资源并且请求的响应时间也会明显加长. 所以本文沿用了大多数负载均衡机制的解决方案, 即仅在有状态会话建立之前由负载均衡策略选择合适的服务节点, 并将该会话中发生的请求都发送至该节点进行处理. 而且由于该方法的机制相对独立, 所以既可以适用于服务最终面对的终端用户, 也可以为多层服务架构中不同层服务之间的调用提供负载调度.

同时, 本文采用区分服务流 (service flow) 和控制流 (control flow) 的方式将集群视图的动态更新和负载均衡策略的动态调整两种功能引入客户代理. 服务流是指在服务调用过程中, 影响调用过程和结果的操作流程; 而控制流则相反, 它是在调用过程之外用于触发和进行附加操作的流程. 这样客户代理在服务调用过程之外还可以进行一些附加操作, 如更新集群视图, 以实现动态功能.

如图 2 所示, 本文的客户端负载均衡机制分为客户代理和服务器端两部分.

2.1 服务器端体系结构

服务器端体系结构为集群节点的动态加入与退出、集群的动态调整等关键技术提供了核心支持, 其各组成部分的功能如下:

• 群组通信设施 (group communication infrastructure) 是一种可靠的组通信中间件, 它允许开发人员配置各种不同的协议栈来进行组播通讯. 同时, 群组通信设施能够通过通信探测的方式及时发现组播列表中成员的加入、离开和丢失, 并可以通过回调接口通知上层应用.

• 服务器端视图管理器 (server-side view manager) 动态获取和维护了当前集群中各节点的状态信息. 在本文的框架中, 视图管理器分为服务器端和客户端两种, 它们之间的协作方式将在 3.1 小节和 3.2 小节进行说明.

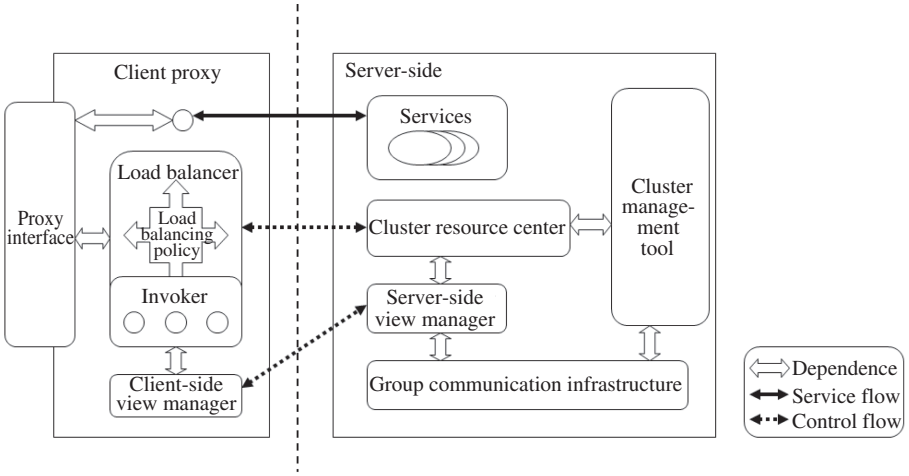

图 2 体系结构

Figure 2 The architecture of the cluster

• 集群资源管理中心 (cluster resource center) 管理了当前集群中所有集群化的服务实例的相关信息, 包括实例特定的负载均衡策略等.

• 集群管理工具 (cluster management tool) 为集群管理员以及自适应系统提供了控制接口. 例如, 通过监视集群的负载情况, 管理员或自适应系统可以使用管理工具动态调整应用在运行中的负载设置 (如负载均衡策略等), 从而更好地实现负载的合理分配, 有效地提高系统的性能.

2.2 客户端代理

客户代理的主要变化在于它具备了从服务器端获取动态更新的功能. 该功能主要由以下各组成部分提供:

• 客户端视图管理器 (client-side view manager) 从服务器端通过控制流获取集群视图的改变信息, 其主要作用将在 3.2 小节进行说明.

• 负载均衡器 (load balancer) 负责管理负载均衡策略实例以及相应的一组服务实例调用器. 在用户需要发送请求时, 负载均衡器调用负载均衡策略实例从众多服务实例调用器中选择出一个合适的, 并将客户调用交由该调用器完成. 同时, 负载均衡器负责发现和恢复由于集群视图突然变化 (节点意外退出) 而引起的调用失效. 负载均衡策略的动态调整也将由负载均衡器完成, 相关内容将在 3.3 小节进行说明.

• 代理接口 (proxy interface) 是客户唯一可见的业务逻辑接口, 它与传统的服务调用过程中使用的业务接口没有任何区别, 只不过它在屏蔽远程调用过程的同时也向最终用户隐藏了集群的各种信息, 实现了客户透明.

3 关键流程

3.1 集群节点的动态加入与退出

集群节点的动态加入与退出是指在不中止系统服务的前提下, 计算节点能够根据系统负载等的需要动态的加入或退出集群. 本文使用可靠的群组通信设施来检测网络层节点的变化: 当新的节点加入

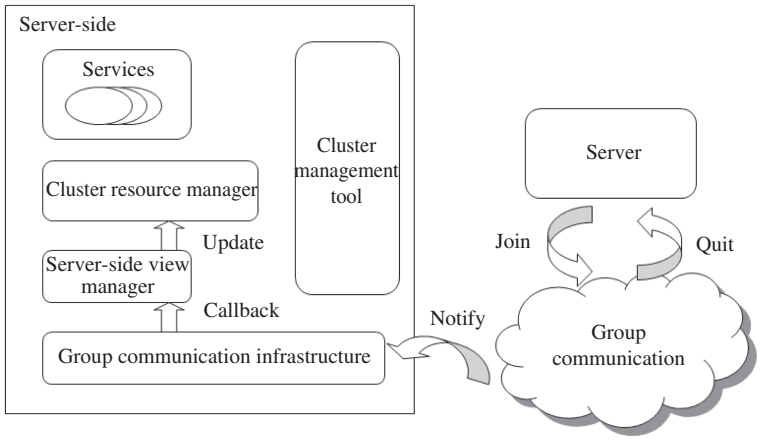

图 3　集群节点的动态变化

Figure 3　The dynamic changes of the cluster

或旧有的节点退出 (无论主动退出还是异常退出) 组播通讯列表时, 群组通信设施会触发相应的回调方法, 将集群视图的最新变化通知上层的服务器端视图管理器. 由于集群视图的变化意味着服务资源也在发生着改变, 集群资源管理中心同样具有监控集群视图改变的能力, 只不过它关注的粒度更细, 具体到特定的服务资源. 群组通信设施、服务器端视图管理器和集群资源管理中心共同维护了集群中视图和资源的一致性. 它们之间的协作方式如图 3 所示.

当有新节点加入时, 它会主动加入当前集群的组播通讯列表, 因此其他节点的群组通信设施会及时接收到新节点加入的消息. 群组通信设施通过调用回调方法, 使得服务器端视图管理器能够及时更新本节点的集群视图信息. 而当新节点的服务实例启动时, 配置为集群资源的服务在集群化的过程中会将其调用器 (封装成资源的形式) 加入该节点的集群资源管理中心. 同时, 集群中其他节点的集群资源管理中心都会接收到资源改变的消息, 并及时更新自身所维护的资源列表.

而当集群中有节点退出时, 其他节点的群组通信设施同样会接收到节点退出的消息. 群组通信设施通过调用回调方法, 使得服务器端视图管理器更新本地的集群视图信息. 但对于集群资源管理中心, 则需要及时更新每一个含有退出节点服务实例调用器的资源. 在遍历资源中心的过程中, 系统并没有暂停服务, 这样就有可能在遍历没有完成的情况下, 新的客户端获得已经失效的服务调用器. 在这种情况下, 将由客户代理的负载均衡器对可能发生的失效调用进行恢复.

3.2　客户端集群视图的动态更新

服务器端集群视图的及时更新可以帮助新的客户更容易地访问到有效的资源, 但是对于集群视图改变之前已经存在的客户来说, 无法帮助他们发现新的资源和识别已经失效的资源. 因此, 客户端的集群视图也需要能够及时进行更新, 以便于负载均衡器能够做出正确的决策.

为了实现客户端集群视图的动态更新, 本文使用控制流的方式建立客户端视图管理器与服务器端视图管理器之间的联系. 如 3.1 小节所述, 服务器端视图管理器中的信息都是从群组通信设施中即时获得的, 尽管不同的集群节点在某一个瞬间时刻所维护的集群视图可能会有所不同, 但是因为群组通信的可靠性, 所以不同节点的集群视图在更新时间上只会有非常短的时间差, 所以各个节点上维护的视图信息在绝大多数时间内是保持一致的. 因此客户端集群视图管理器只需要定时选取任意一个集群

节点进行集群视图的更新. 如果集群视图确实发生了变化, 客户端视图管理器将通过回调方法通知负载均衡器进行进一步的处理, 例如, 从待选列表中去除已经失效的服务实例调用器.

可以看出, 客户端集群视图的动态更新势必会带来一定的网络传输开销, 因此本文提出的控制流并不局限于一种特定的实现技术. 当集群视图变动频繁时, 客户代理可以建立一条独立的网络连接进行集群视图的快速更新; 在集群视图并不频繁改变的情况下, 客户代理也可以在发送业务请求时采用信息捎带的方式在每一次或者每几次业务请求时更新集群信息, 从而通过降低集群视图更新频率来提高系统性能. 在更新策略上, 控制流也需要同时能够支持 "推" 和 "拉" 两种更新方式, 以满足不同应用场景下的需要. 同时, 客户端对集群视图的更新频率、更新策略等都可以预先进行配置, 也可以通过集群管理工具在运行时刻进行动态修改.

3.3 负载均衡策略的动态调整

负载均衡策略及参数的修改是管理员通过集群管理工具在运行时刻完成的. 负载均衡策略的动态调整同样分为服务器端和客户端两部分.

在服务器端, 当节点的集群资源管理中心收到集群管理工具修改负载均衡策略的命令时, 它会替换该服务资源的负载均衡策略及其相关参数. 同时, 修改了负载均衡策略的节点也会向其他节点发送组播消息, 使得其他节点的资源管理中心也能进行相应的更新.

客户端负载均衡策略的动态更新与集群视图的动态更新类似, 也是使用了控制流的方式使得客户端的负载均衡器与服务器端的集群资源管理中心能够定时同步信息. 同样地, 客户端对负载均衡策略的更新频率、更新策略都可以预先进行配置并通过集群管理工具进行动态修改.

4 实例分析

PKUAS (PeKing University Application Server)[10,11] 是北京大学信息科学技术学院软件研究所自主开发的 JEE 应用服务器, 目前已经在 OW2 平台 3) 上使用 GPL 协议开源. 其最新的稳定版本 PKUAS OSGI 兼容 JEE 5 的主要规范. 我们选择在 PKUAS OSGI 上实现本文提出的动态客户端负载均衡机制, 其中一些关键的实现技术如下.

4.1 控制流的实现

PKUAS OSGI 的 EJB 集群使用截取器技术将控制流的信息附加在业务请求和应答消息上, 如图 4 所示. 在每次业务调用时, 调用请求在客户代理中要通过一组请求预处理截取器, 在这个过程中客户端视图管理器和负载均衡器会通过特定的截取器将各自的更新请求与业务请求编码在一起, 然后通过网络发送给服务器. 服务器在接收到请求消息之后, 同样首先交由一组预处理截取器处理, 服务器端视图管理器和集群资源管理中心会通过截取器将附加在业务请求上的控制流信息剥离出来, 而最终的业务请求则交由相应的 EJB 实例进行处理.

应答信息返回也同样需要经历一个类似的过程, 服务器端通过截取器在应答信息中附加更新消息, 客户端则通过截取器获取更新消息.

3) http://ow2.org/

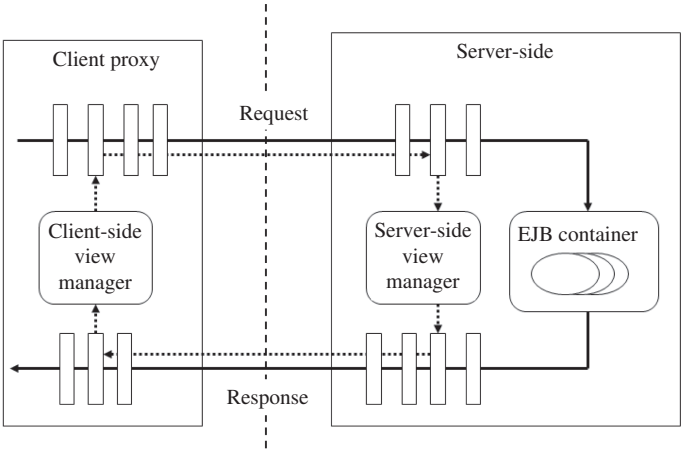

图 4　基于截取器机制的控制流实现

Figure 4　The control flow via the interceptor mechanism

4.2　群组通讯设施的实现

PKUAS 中的群组通讯设施使用了开源组通信中间件 JGroups[12]. JGroups 是用 Java 语言编写的可靠组播通讯工具集. 在多对多通信十分频繁的情况下, 基于 TCP 协议的 JGroups 表现出良好的性能与伸缩性 [13]. 经过分析, 从功能和性能上, JGroups 都满足本文提出的机制对组通信设施的需求, 因此 PKUAS OSGI 选用其作为群组通信设施.

4.3　负载均衡策略的实现

目前 PKUAS EJB 集群的负载均衡体系中, 共实现了 4 种负载均衡策略, 它们分别是:

• RoundRobin 是依次从可用节点列表中选择下一个节点, 第一个节点则随机选择.

• Random 是每次从节点列表中随机选择一个节点.

• FirstAvailable 指选择节点列表中第一个可用的节点.

• StaticWeighted 是指对每个节点都指定一个静态权值 W, 任意一个节点在每次选择时被选中的概率为该节点的静态权值除以所有节点的权值之和.

5　实验分析

5.1　实验设计与实验环境

本文模拟电子商务结算系统作为测试用例. 集群中运行的已经集群化的 EJB 模拟了结算中心系统中的计价功能. 大量客户端模拟了进行网络购物的客户人群, 它们会随机选取虚拟商品发送到结算中心, 结算中心在收到请求之后会随机加入会员折扣等其他因素进行价格计算. 每个客户端会随机进行一次到多次的购物行为, 然后退出. 这样的实验设定以求直观反映集群动态变化对长期存在的客户代理的影响.

集群实验环境配置如下: 4 台 PC 机作为集群服务器, CPU 与内存分别为 P4 2.8 GB 和 1 GB. 实

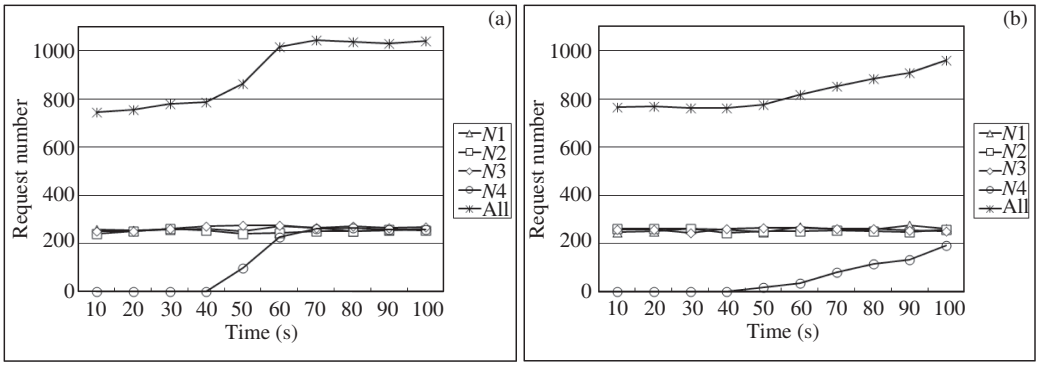

图 5 节点动态加入对系统性能的影响

Figure 5 The changes in performance when a new node joins in the cluster. (a)Dynamic client; (b)static client

验用 PC 都安装了 Windows XP SP3 操作系统, JDK 版本均为 build 1.6.0_07-b03. 服务器节点间通过 1000 Mbps 以太网互联.

客户端程序运行在另外一台独立的相同配置的 PC 上, 与 4 台服务器通过 1000 Mbps 以太网连接. 实验数据全部由客户端程序进行记录, 主要记录各个节点单位时间内成功处理请求的数量.

5.2 集群节点的动态加入

本项实验记录了在应用运行过程中动态加入计算节点前后系统性能的变化. 应用负载均衡策略始终选用 RoundRobin, 集群视图更新频率设置为每次业务调用时进行更新. 实验过程为: 首先启动 $N1$, $N2$ 和 $N3$ 节点, 待 3 个节点单位时间内处理请求的数量稳定之后, 启动 $N4$ 节点并使其动态加入集群. 实验结果中相邻记录点之间的时间间隔设置为 10 s, 用户并发数为 100. 实验结果如图 5(a) 所示. 从实验结果可以看出, 在集群中动态加入的节点能够迅速被客户端负载均衡器所发现: 在加入集群之后, $N4$ 的负载很快就上升到与 $N1$, $N2$ 和 $N3$ 相同的水平. 因此, 整个集群单位时间内的请求处理量在 $N4$ 加入之后迅速提升 30% 左右.

作为对比实验, 本文在使用静态客户端负载均衡机制的系统中重复了上述实验过程, 系统配置和实验设置完全相同. 实验结果如图 5(b) 所示. 由于已经存在的客户端负载均衡器无法感知到新节点 $N4$ 的加入, 而只有新产生的客户端才有可能选择对节点 $N4$ 上服务进行访问, 所以在加入新结点之后集群系统的性能改善的较为缓慢.

5.3 集群节点的动态退出

本项实验记录了在应用运行过程中动态退出计算节点, 系统整体性能的变化过程. 应用负载均衡策略选用 Random, 集群视图更新频率设置为每次业务调用时进行更新. 实验过程为: 启动所有节点进行服务, 待节点的请求处理量稳定之后, 提前通知负载均衡机制 $N4$ 节点将退出集群, 并在通知后 10 秒时将 $N4$ 节点退出集群. 实验结果中相邻记录点之间的间隔设置为 10 s, 用户并发数为 100. 实验结果如图 6(a) 所示. 从实验结果可以看出, 集群节点的退出可以在短时间内对集群的负载能力产生较大的影响 (时间为 40 s 时开始), 但是由于客户端能够及时获得集群视图的变化信息, 所以负载从客户代理被转发到了其余节点上. 同时, 由于客户端视图的及时更新, 在 100 个客户端中, 共有 2 个因为

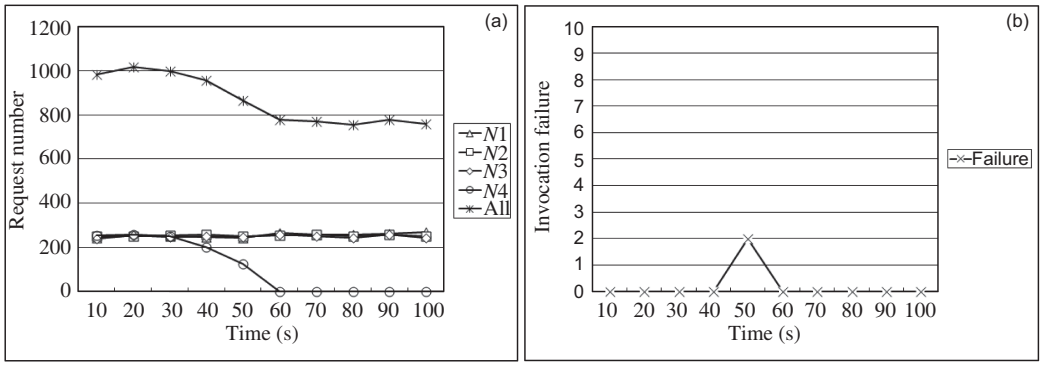

图 6　动态客户端系统中节点动态退出对系统性能的影响

Figure 6　The changes in performance when a node quits from the cluster. (a)Request number; (b)invocation failure

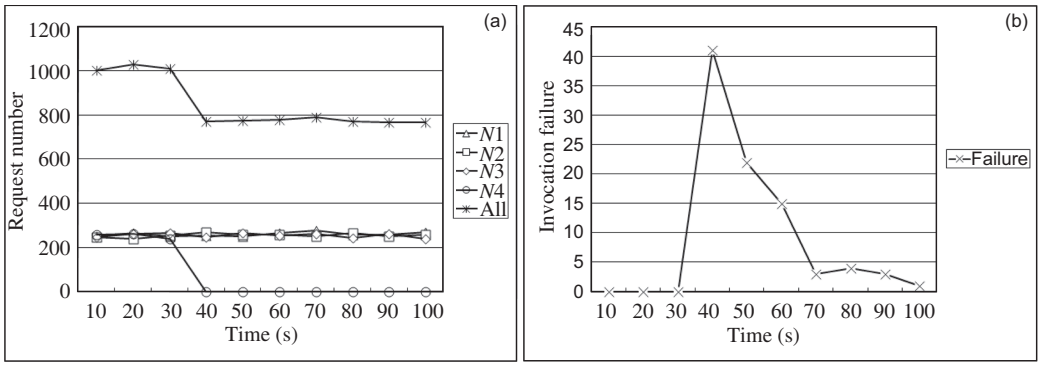

图 7　静态客户端系统中节点动态退出对系统性能的影响

Figure 7　The changes in performance when a node quits from the cluster. (a)Request number; (b)invocation failure

$N4$ 节点退出而发生了失效调用 (需要相对较长的时间进行错误恢复), 而其余 98 个则避免了将请求发送到已失效的节点, 如图 6(b) 所示.

作为对比实验, 本文在使用静态客户端负载均衡机制的系统中重复了上述实验过程, 由于无法通知客户端负载均衡机制, 该实验中节点将直接退出集群系统, 系统配置则与之前完全相同. 实验结果如图 7(a) 和 (b) 所示. 由于已经存在的客户端负载均衡器无法感知到节点 $N4$ 的失效, 所以有 88 个客户端发生了失效调用, 而其余 12 个客户端则是因为在退出系统之前没有选择访问 $N4$ 节点上的服务而避免了失效调用的发生.

5.4　负载均衡策略的动态修改

本项实验记录了在应用运行过程中动态修改节点负载均衡策略对系统整体性能的影响. 实验开始时应用负载均衡策略选用 FirstAvalilable, 集群视图更新频率设置为每次业务调用时进行更新. 实验过程为: 启动所有节点进行服务, 待节点的请求处理量稳定之后, 注意到 $N1$ 和 $N2$ 节点处于高负载状态下, 而 $N3$ 和 $N4$ 节点则始终是低负载状态. 集群管理员通过集群管理工具动态修改系统的负载均衡算法为 StaticWeighted, 并将所有的节点的权重都设置为 2.5. 实验结果中相邻记录点之间的时间间隔为 10 s, 用户并发数为 100. 实验结果如图 8 所示.

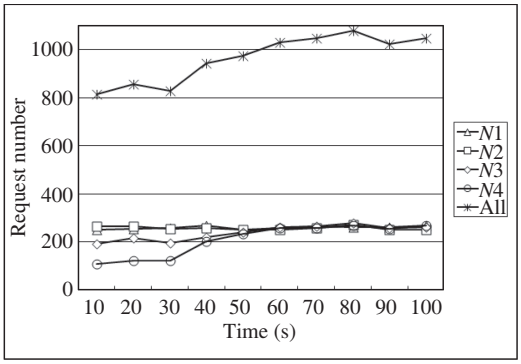

图 8 负载均衡策略动态修改对系统性能的影响

Figure 8 The changes in performance when the load balancing policy is changed

由于使用 FirstAvalilable 负载均衡算法, 在系统高负载的情况下, 大量客户端负载均衡器都选择访问 $N1$ 节点, 只有当访问超时或者发生错误时才依次选择其他节点, 这导致了系统在实验开始时的负载失衡. 当动态修改负载均衡策略后 (图 8 中 40 s 之后), 在较短的时间内更多的负载从客户端负载均衡器被分发到 $N3$ 和 $N4$ 节点, 集群整体的性能由于负载得到了更好地分配而得到了明显的提高 (All 曲线出现上升).

而对于采用静态的客户端负载均衡机制的系统而言, 如果要修改负载均衡算法及配置参数, 则整个集群系统需要重新启动之后才能生效, 因此无法在保证在不中止服务的前提下解决系统面临的负载失衡问题.

6 相关工作

客户端负载均衡技术并非是一个新出现的概念 [14]. 网景公司 (Netscape Communication) 曾提出了一种基于客户端的请求调度方法 [15]. 当用户访问网景的主页 (http://www.netscape.com) 时, 导航器 (navigator) 在 1 和服务器总数之间产生一个随机数 i 并将用户请求定向到服务器 wwwi.net-scape.com. 这种方法局限性很强且基本不具有可伸缩性, 主要是在企业内部网中发挥一定的作用.

Yoshikawa 等在他们的 WebOS 项目[16,17] 中提出了一种动态的客户端负载均衡机制 smart client[18]. 这种机制依赖于 Java Applet 技术, 主要关注于客户透明的代理方式, 并在 Telnet, FTP 等的客户端应用程序中进行了实现. 本文的客户端负载均衡机制基于使用更加广泛的客户代理技术, 所以并不需要过多关注怎样解决客户端透明的问题, 而更加注重服务器端集群技术对客户端动态负载均衡机制的支持, 如集群节点的动态调整、负载均衡机制的动态重配置, 这些问题在 Yoshikawa 等的文章中并没有进行详细说明.

Sewook 等 [19] 详细分析了现有的多种负载均衡机制如集中式的负载均衡机制、DNS 负载均衡方法等在云架构环境下存在的可伸缩性不强的问题, 并提出了一种客户端负载均衡方法. 该方法主要关注于客户端负载均衡方法的可扩展性, 即支持随着集群节点的大规模变化集群性能相应的改变. 但是该方法并不支持对客户端负载均衡机制的动态更新, 已经存在的客户端因此也无法感知集群状态的改变, 这与本文提出的方法有明显的不同.

当前主流的 Jave EE 应用服务器大都选择了使用客户代理作为其负载均衡机制的一种实现方式.

作为最成功的开源应用服务器之一, JBoss[4] 目前能够支持集群视图变化时客户代理中节点列表的动态更新, 但不支持运行时刻替换缺省的负载均衡算法, 而且也无法在运行时刻修改负载均衡算法的配置参数. 而 JOnAS[20] 最新的稳定版本目前尚不支持客户端动态负载均衡机制. BEA WebLogic[5] 是最成功的商业应用服务器之一. 它提供了功能全面的动态集群服务, 支持集群节点的动态加入与退出. 其不足之处在于, 尽管 WebLogic 集群服务支持运行时刻修改一些集群的缺省设置, 但是却需要在应用重新启动之后才能生效, 因此在对集群的配置管理上灵活性不足.

7 结束语

本文提出了一种动态的客户端动态负载均衡机制以解决目前静态的客户端负载均衡机制及其服务器端支撑技术所面临的适应网络变化能力不足的问题. 通过引入分布式的集群视图更新、控制流等技术, 本文在保持客户端负载均衡机制可伸缩性强的基础上, 又为集群节点的动态加入与退出和负载均衡策略的动态重配置提供了有效的支持.

动态的客户端负载均衡机制不仅为负载均衡策略的动态调整提供了支持, 而且也为多种动态负载均衡算法的数据传输提供了支撑技术. 但是本文提出的机制目前尚未对动态负载均衡算法进行深入的研究, 我们希望在将来的工作中能更好的支持这些机制, 以更灵活的方式支持多种动态负载均衡策略.

参考文献

1 Yang F Q, Mei H, Lv J, et al. Some discussion on the development of software technology. Chin J Electron, 2003, 26: 1104–1115 [杨芙清, 梅宏, 吕建, 等. 浅论软件技术发展. 电子学报, 2003, 26: 1104–1115]

2 Mei H, Huang G, Xie T. Internetware: Software paradigm for internet computing. IEEE Comput, 2012, 45: 26–31

3 Abraham K. J2EE clustering. http://www.javaworld.com/jw-02-2001/jw-0223-extremescale.html, 2001

4 Brian S, Galder Z. JBoss application server clustering guide. 2011

5 BEA System Inc. Using WebLogic server clusters. 2007

6 Karve A, Kimbrel T, Pacifici G, et al. Dynamic placement for clustered web applications. In: Proceedings of the 15th international Conference on World Wide Web (WWW '06), New York, 2006. 595–604

7 Tang C, Steinder M, Spreitzer M, et al. A scalable application placement controller for enterprise data centers. In: Proceedings of the 16th international Conference on World Wide Web (WWW '07), New York, 2007. 331–340

8 Marshall P, Keahey K, Freeman T. Elastic site: Using clouds to elastically extend site resources. In: IEEE International Symposium on Cluster Computing and the Grid, Washington, 2010. 43–52

9 Luis M V, Luis R M, Rajkumar B. Dynamically scaling applications in the cloud. SIGCOMM Comput Commun Rev, 2011, 41: 45–52

10 Huang G, Wang Q X, Cao D G, et al. PKUAS: a domain-oriented component operating platform. Chin J Electron, 2002, 30: 1938–1942 [黄罡, 王千祥, 曹东刚, 等. PKUAS：一种面向领域的构件运行支撑平台. 电子学报, 2002, 30: 1938–1942]

11 Meng J. The design and implementation of EJB level dynamic clustering service on application server PKUAS. Master Thesis. Beijing: Peking University. 2008 [孟嘉. 应用服务器 PKUAS 中 EJB 层动态集群服务的设计与实现. 硕士学位论文. 北京: 北京大学. 2008]

12 Bela B. Reliable multicasting with the JGroups toolkit. 2008

13 Abdellatif T, Emmanuel C, Renaud L. Evaluation of a group communication middleware for clustered J2EE application servers. In: On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE, Agia Napa, 2004. 1571–1589

14 Cardellini V, Colajanni M, Yu P S. Dynamic load balancing on web-server systems. IEEE Internet Comput, 1999, 3: 28–39

15 Mosedale D, Foss W, McCool R. Lessons learned administering Netscape' s internet site. IEEE Internet Comput, 1997, 1: 28–35
16 Vahdat A, Anderson T, Dahlin M, et al. WebOS: Operating system services for wide area applications. In: Proc. of the 7th IEEE International Symposium on High Performance Distributed Computing, Chicago, 1998. 52–63
17 Vahdat A, Dahlin M, Eastham P, et al. WebOS: Software support for scalable web services. In: Proc. of the 6th Workshop on Hot Topics in Operating Systems, Cape, Cod, 1997. 106–114
18 Yoshikawa C, Chun B, Eastham P, et al. Using smart clients to build scalable services. In: Proc. USENIX 1997 Annual Technical Conference, Berkeley, 1997. 105–117
19 Sewook W, Huan L. Client-side load balancer using cloud. In: Proceedings of the 2010 ACM Symposium on Applied Computing (SAC '10). New York: ACM, 2010. 399–405
20 The JOnAS Group. Java open application server (JOnAS): architecture and design. 2010

A dynamic client-side load balancing mechanism

WANG ZiYou[1,2], ZHOU MingHui[1,2*] & MEI Hong[1,2]

1 *Software Institute, School of Electronics Engineering and Computer Science, Peking University, Beijing* 100871*, China;*
2 *Key Laboratory of High Confidence Software Technologies* (*Peking University*)*, Ministry of Education, Beijing* 100871*, China*
*E-mail: zhmh@pku.edu.cn

Abstract Web applications' traffic demand fluctuates widely and usually unpredictably. The common practice of provisioning a fixed capacity would either result in unsatisfied customers (when the resources allocated to a specific customer are too low) or waste valuable capital investment (when the resources allocated to a specific customer are too much). By leveraging an infrastructure cloud's on-demand, pay-per-use capabilities, we finally can match the system's capacity with the demand in real time. Although the static client-side load balancing mechanism has been used by many clusters, it does not suit the cluster which needs to change nodes and load balancing policies in runtime. This paper proposes a dynamic client-side load balancing mechanism. By integrating technologies such as distributed cluster-view maintenance and control flow, this mechanism not only has good scalability but also introduces many dynamic features into the client-side load balancing, providing effective support for the nodes' dynamic join and exit as well as the adaptive adjustment of the load balancing policies. Since the framework has been implemented in an open-source JEE application server named PKUAS, this paper also describes some key implementation features and analyzes the evaluation results of the experiments to show the effectiveness of the mechanism from different aspects.

Keywords cloud, dynamic cluster, client-side load balancing, cluster view, client proxy

WANG ZiYou was born in 1983. He is currently working toward the Ph.D. degree at the School of Electronics Engineering and Computer Science, Peking University, Beijing. His research interests are in the areas of middleware clustering and overload management.

MEI Hong is a full Professor of School of Electronics Engineering and Computer Science, Peking University. He received the Ph.D. degree in computer science from Shanghai Jiao Tong University in 1992. His current research interests are in the area of software engineering, software reuse and software component technology, and distributed object technologies. He is a member of the Expert Committee for Information Technology of National High-Tech Research and Development 863 Program of China, a chief scientist of National Basic Research 973 Program of China, and the director of Technical Committee on System Software (TCSS) of China Computer Federation (CCF). He is also a fellow of CCF.

ZHOU MingHui was born in 1974. She received the Ph.D. degree in 2002 from the National University of Defense Technology, Changsha, Hunan. Currently, she is an Associate Professor at Peking University. Her research interests include empirical software engineering, mining software repositories, open source software development, and middleware. Dr. Zhou is a member of ACM and China Computer Federation (CCF).