DartCache:一个基于哈希表的分布式 Cache 系统

谢骋超　陈华钧　张　宇
(浙江大学计算机科学与技术学院　杭州 310027)

摘　要　随着现代企业应用开发的规模越来越大,系统的性能与可伸缩性对系统的成败起着越来越重要的作用。在整个分层的企业级应用架构中,数据库层的访问速度通常会成为整个系统的瓶颈。采用 Cache 技术将已访问的数据缓存在应用层,从而减少数据库访问的负载量,可以使系统的性能与可伸缩性得到根本性的提高。在吸取了前人 Cache 开发经验的基础上,我们设计了 DartCache,它是一套基于哈希表的分布式 Cache 系统。经实验证明,DartCache 使系统的性能与可伸缩性都得到了根本性的提高。

关键词　分布式,Cache,哈希表,性能,可伸缩性

DartCache: a HashMap-Based Distributed Cache

XIE Cheng-Chao　CHEN Hua-Jun　ZHANG Yu
(College of Computer Science, Zhejiang University, Hangzhou 310027)

Abstract　With the development of modern enterprise application, the scalability and performance issue begins to play a more and more important role. Within the layered enterprise application architecture, database is usually the bottleneck of system. Cache technology can significantly improve the system performance and scalability by caching the data in application layer. Based on the experience of previous Cache development, we design DartCache, a HashMap based distributed cache system. The experiments have shown a significant improvement of performance and scalability with the implementation of DartCache.

Keywords　Distributed, Cache, HashMap, Performance, Scalability

1　引言

在现代企业级分层架构里,数据库通常会成为整个系统性能和可伸缩性的瓶颈。采用 Cache 技术将已访问的数据缓存在应用层,从而减少数据库访问的负载量,可以使系统的性能与可伸缩性得到根本性的提高。

图 1 是一个典型的企业级应用的分层架构图。

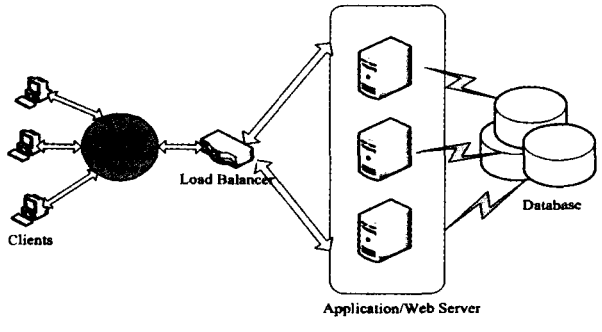

图 1　典型的企业级应用分层架构

为了提高系统的可伸缩性、稳定性和容错性,通常的大型应用里的 Web 服务器都会分布在集群上,由统一的负载均衡器响应请求,将这个请求分发到负载较轻的 Web 服务器。在 Web 服务器的后端则是我们的数据库,几乎每次的业务请示都会让系统去查询或更新后端数据库的数据库。从实践的角度,数据库存在着以下几个特点:

1)数据库与 Web/Application 服务器的距离很遥远。数据库运行在另一台机器的不同进程里,每次数据传输都需要将数据放在网络上进行序列化与反序列化。就像 Peter Deutsch 所提出的网络的 8 个谬论[2]里,网络数据传输的代价是不能忽略的。也许需要穿越好多层的防火墙,也许数据库服务器所在网络正在承受着巨大的负担。

2)数据库很难像 Web 服务器或应用服务器那样通过集群轻松地提高可伸缩性。在 Web 服务器或应用服务器的集群里,每台机器几乎都不需要保存状态(除了少量的 Session 数据),因此每台机器之间只需要进行少量的数据传输。当一个请求来时,Web 服务器不需要(或少量)跟集群中的其它机器进行交互就能很快地响应请求。然而,数据库的职责却是保存状态,而且这个状态是保存在硬盘(硬盘的速度要比内存慢上好几个数量级)上的,每一次的增删改请求都会导致数据库上数据的变化,也意味着每一次更新都会导致集群上数据库服务器的同步,而这个同步的代价是很大的。而且,我们还要解决更复杂的问题:数据同步的冲突问题。假如有两个请求同时修改了数据库某张表的某条记录,如果只有一台机器,那么数据库内部的锁机制会自动将这条记录保护起来;但是当有好几台机器时,如果修改的结果在不同的机器上,那么数据库的锁保护机制也失效了。虽然有 Oracle 9i RAC 等产品帮我们解决了数据库在集群上使用的问题,但这样的代价是非常大的。

3)数据库对于大数据量的分类、查询、计算有着很好的优

谢骋超　硕士研究生;陈华钧　讲师;张　宇　博士研究生。

化和性能。成熟的数据库产品如 Oracle、DB2、SQL Server 都是经过了几年到几十年的考验，对大批量数据的分类、查询操作等算法做了大量的优化。在这种操作的效率上它甚至会比一般内存中的对象更快。

数据库的这些特点给我们带来了严峻的问题：性能与可伸缩性。很多大型的企业级应用的性能瓶颈都存在于对数据库的访问上。数据库的第 1 个特点，使我们想到了既然数据库离应用程序的距离太遥远，那么为什么不将应用程序与数据库的距离拉近点？这就是 Ted Neward 在 Effective Enterprise Java[3] 里提出的将数据与处理过程放在一起。

Cache 的意义在于将大量的数据拉到应用程序层，所有的访问在应用程序层拦截掉了。当我们接受相同的请求时，我们会把上一次执行结果缓存在应用程序层，下次的请求不需要再去访问数据库了。这样的结果不仅大大提高了应用程序的速度，也给数据库服务器大减负，使数据库的访问性能大大提高。

图 2 是系统使用 Cache 之后的系统架构图。

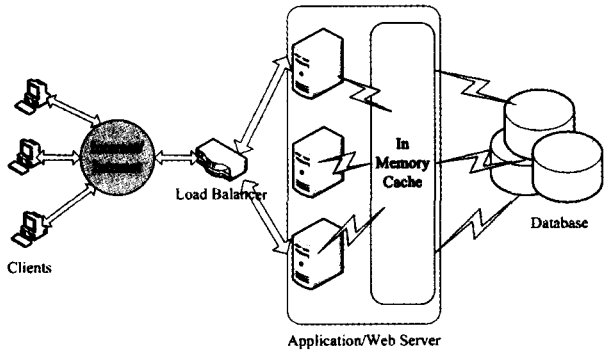

图 2　使用 Cache 后的企业级应用分层架构

2　相关工作

Cache 的思想并不是新的，全世界有很多人都在想办法提高应用程序的性能，都在开发自己的 Cache 产品。下面分析一下几个典型的 Cache 产品及它们的优缺点。

2.1　EhCache

EhCache[4] 是一个很简单的 Cache 系统，是一个用 Java 开发的开源项目。它对 Java 的 HashMap 进行了扩展，使 Cache 的数据以对象的形式放在 HashMap 里，但是比 HashMap 提供了更好的同步机制和数据溢出机制及过期数据淘汰机制。这意味着当 Cache 里的数据达到一定度时，它会自动将溢出数据存储在硬盘上。EhCache 的优点也正是在于它的简单性。正是由于它的简单和小巧使它在很多小型的应用里得到了广泛的使用。

但是 EhCache 只能使用在小型系统里，因为它有一个致命的弱点：它没有提供任何在集群上使用的同步机制。这意味着当 EhCache 被使用在集群上时，每个 Web/Application 上的 Cache 可能会不一致，从而导致最后数据访问的冲突。

2.2　OSCache

OSCache[5] 是 OpenSymphony 开发的一个开源的 Cache 产品，其基本实现功能与 EhCache 类似。但是它在此之上做了很多的扩展，尤其是对 Web 应用中部分页面的 Cache，使用少量 OSCache 的标签后，用户可以很轻松地实现对页面数据的 Cache。

它还提供了使系统在集群上运行的扩展。它利用 JMS 或者 JGroups，使集群上其它节点的 Cache 数据自动失效。但是，OSCache 所提供的 cluster 功能是相当有限的，无法让 Cache 中的数据在各个节点之间复制和分布，也无法让 Cache 的功能在整个集群上得到使用。它唯一做的只是利用 Cache 失效功能避免了集群上不同节点数据同时修改时带来的 Cache 数据冲突。

2.3　JBoss Cache

JBoss Cache[6] 是 JBoss 推出的开源的 Cache 产品，其实现功能与前两种 Cache 稍有不同，它的 Cache 结构是树状的，而每个树状节点下面才是某个 Cache 的 HashMap。JBoss 的这种设计，其目的是想通过 Cache 数据在树状节点的分布，来减少数据在集群节点间复制的代价。但是事实上，这种设计方案并没有带来太多的性能改进，反而使系统的 API 变得相当复杂了。

JBoss 提供了 Cache 数据在 cluster 节点复制的功能，同时提供了完整的 Cache 事务处理机制，使 Cache 数据在某个事务里可以 rollback。它还提供了完整的 AOP 功能，以简化整个 Cache 的 API，使开发者在开发时不需要调用 Cache 的 API，就能完成 Cache 数据的 get 和 put，但是它仍然存在着以下缺点：

①Cache 数据在 cluster 上的分布功能仍然是有限的，只能提供数据的复制，而没有提供更优的算法，使数据节点在 Cache 上更有效地分布，从而最大化地利用 cluster 的数据资源。

②它提供的 AOP 功能虽然简化了 Cache 的 API，但是会对系统的性能带来一定的影响。因为它的 Cache 同步机制是发生在 Cache 对象的每次 set 操作之后的，如果 Cache 要 set 10 次，则它必须在 cluster 间发生 10 次同步，这样的性能代价是很大的。

2.4　Tangosol Coherence

Tangosol Coherence[7] 是目前最成熟的基于 cluster 的 Cache 产品。它给用户提供了各种不同 Cache 的选择方案，从 local cache 到 replicated cache，再到 distributed cache。它提供了强大的 Cache 功能，强大的 Cache 分布与复制功能使 Cache 可以在 cluster 中自由地复制、分布。它的负载均衡与错误恢复功能也足以保证系统的可靠性。而它提供的 write through、read through 和 near cache 等功能更使系统的功能变得强大，但 tangosol Coherence 仍然存在着以下缺点：

①它是一个商业软件，4999 美元 CPU 的价格足以吓跑好多客户。

②它没有提供良好的 AOP 功能，设计系统时也没有考虑 AOP 功能，因此所有的 Cache 操作都要用 API 来完成。尽管这个功能可以由第三方包装，但是由于不能从底层开始设想 AOP 功能，因此这些包装都存在一些问题。

③它没有提供集群中 Cache 数据更新时的异步更新机制。

3　DartCache 的基本特点与架构

3.1　CartCache 的基本特点

DartCache 在总结上以上各种 Cache 产品的功能和设计理念之后，做了改进和调整。在功能上至少有以下特点：

1）它利用现有的 Cache 解决方案来完成本地 Cache 功能。现有的 EhCache、OSCache 本地 Cache 模块都可以作为

插件，接入系统中作为本地 Cache 机制。

2）利用 P2P 数据传输及集群节点发现、协调功能，实现 Cache 的复制、分布、同步、负载均衡及失败恢复等功能。

3）利用 AOP 简化整个系统的设计及 Cache 接口的设计。

4）利用多线程数据同步算法，使集群数据在 cluster 上的同步代价最小化。

3.2 DartCache 的外部架构

图 3 是 DartCache 的外部架构图，标明了 DartCache 在企业级应用里所处的地位。

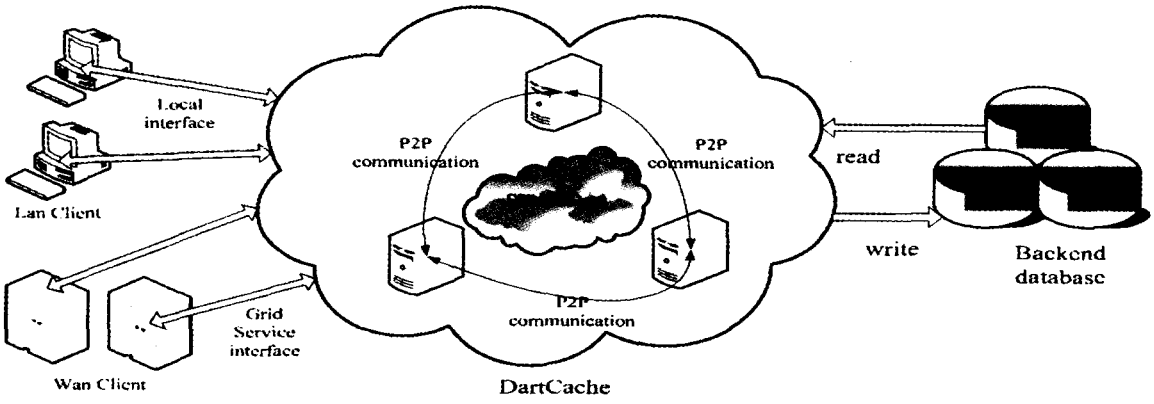

图 3 DartCache 外部架构图

DartCache 分布在企业级应用的应用服务器（或 Web 服务器层）的集群节点上。它们通常会与相应的应用程序共享同一进程，这使应用程序与 DartCache 之间数据交互的代价非常小。

同时，DartCache 分布在集群的不同节点上，这些节点之间也存在着交互，通过这些交互才能实现数据的分布、负载均衡、失败恢复等。它们之间的交互采用的是 P2P 的传输协议，以保证每一台服务器都能独立地响应用户请求。我们使用 JGroups[8] 实现 P2P 传输。

3.3 DartCache 的内部模块组成

图 4 是 DartCache 的内部模块组成图。

Cache Interface 是所有开发者调用 Cache 的接口，它的 API 很简单，只有简单的 get 和 put 方法及一些辅助操作。所有的复杂性都被封装在这个 API 后面了。

Cache AOP 模块则使开发者的负担更轻了，用户不需要调用 Cache 接口就可以完成 Cache 功能。

在 Cache 的后台模块里，Cache Dictionary 起了至关重要的角色，它的结构有点类似于 HashMap，但是它没有真正存放 Cache 的数据，而只存放了 Cache 的一些信息，如 Cache 的节点分布信息，Cache 的访问数据等。

Local Cache 是真正存放本地 Cache 的地方。它提供了一个 Local Cache interface，主要用于 cluster 节点之间的调用。

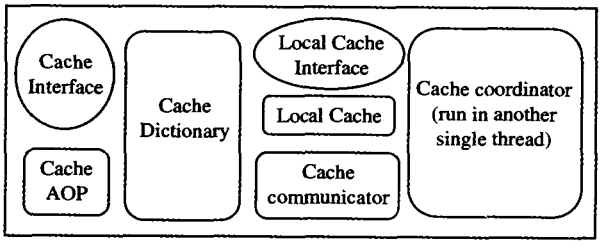

图 4 DartCache 模块组成

Cache communicator 提供了 Cache 数据与 cluster 中的其它节点的交互。

Cache Coordinator 则是 cluster 中其它节点与这个 Cache 交互的接口。它运行在单独的线程里，监听访问的请求，并对它作出响应。

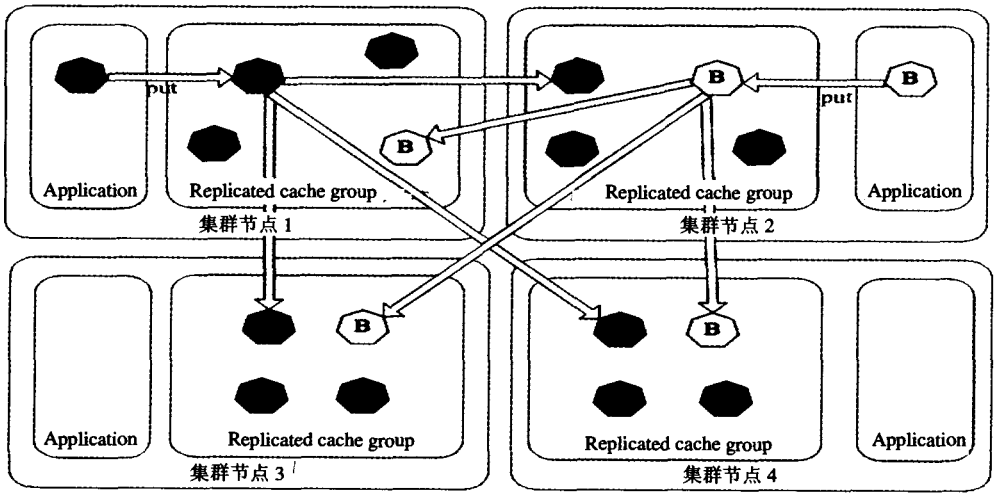

图 5 Replicated Cache 数据更新的过程

4 DartCache 在集群上的数据分布与同步

Cache 数据在集群上的分布是必然的，它保证了 Cache 数据在集群上的正确性，提高了系统的可伸缩性，同时解决了 Cache 数据的负载均衡、失败恢复等问题。但是它并不总是能够提高系统的性能，相反由于在 cluster 中的数据传播时的代价，当系统的访问量小的时候，它反而会使系统的性能降低。因此，系统 Cache 的分布功能一定要谨慎。

我们在系统中实现了两种类型的分布式 Cache：第一种是 Replicated Cache，这与 JBoss Cache 里提供的 cluster Cache 功能类似；另一种是 Distributed Cache，它提供了强大的数据分布算法，使 Cache 数据在整个集群中可以有效地分布，同时保证了系统的可靠性。

4.1 Replicated Cache

Replicated Cache 是指集群上的每个节点所包含的 Cache 内容是一致的。这意味着当某个集群节点的数据改变后，它将广播给集群上的其它节点，使每个节点的数据都会更新成新改变的数据。它的数据更新如图 5 所示。

但是，当我们要从 Cache 中读取数据时，它的操作则变得轻松了。因为每个 cluster 节点上的数据是完全相同的，对任何一个节点数据的读取都会马上从这个节点上得到响应。图 6 是读取 Replicated Cache 上某个数据的示意图。

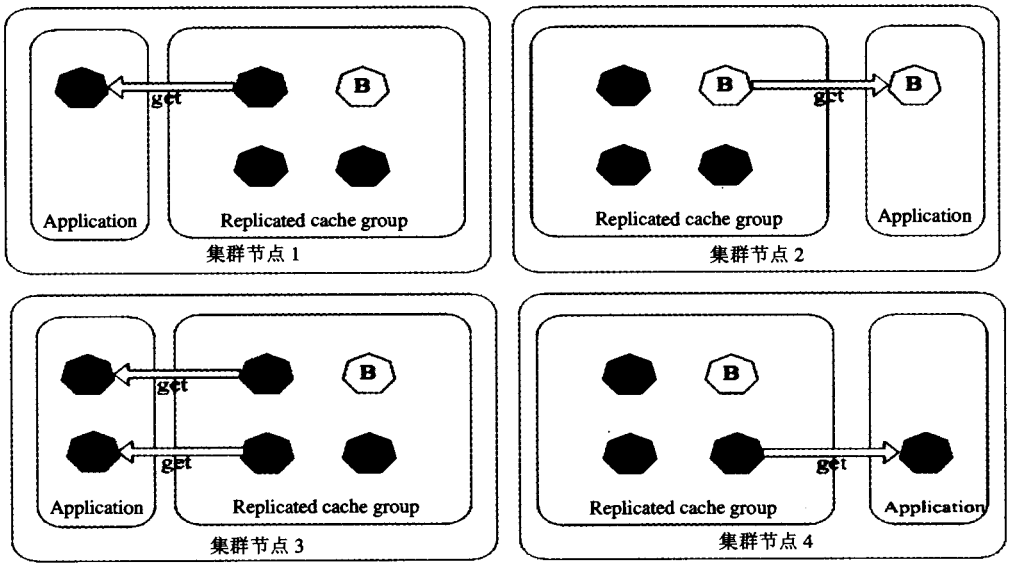

图 6 Replicated Cache 数据获取的过程

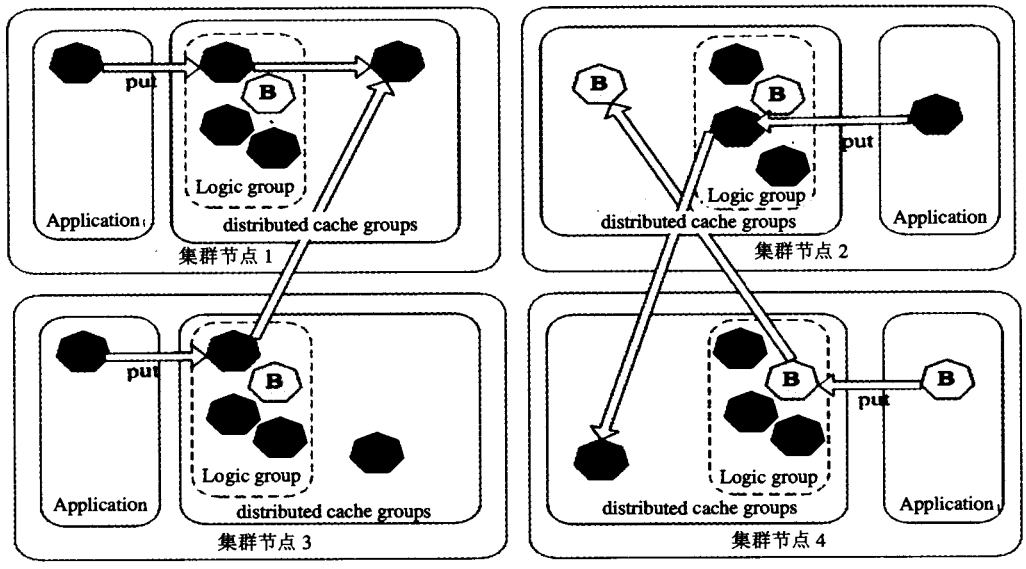

图 7 Distributed Cache 数据更新的过程

Replicated Cache 具有以下几个特点：

①它适用于经常读取而不经常更新的 Cache 数据，因为每次 Cache 数据更新都会引起数据在每个集群的传播，但每次读取不需要经过任何网络传播。

②它具有很强的可靠性，并可使负载均衡和失败恢复变得很简单。由于每个节点的数据都是一致，任何一个节点都可以被其它节点所替换。

③它没有充分利用集群上的内存资源。由于集群上数据的一致性，使得所有集群节点存放的数据内容与一个节点的数据内容是相同的，因此整个集群的内存总数本来有 $n * m$，但却只能存入 m 的数据量。

4.2 Distributed Cache

Distributed Cache 是指将 Cache 上的数据分布在集群的节点上，每个节点存放不同的数据，但又将备份数据保存在集群的另一节点上，这样既保证了数据的可靠性，也保证了数据不存在单点失败。

当数据 Distributed Cache 的数据需要更新时，我们不需要将数据的内容广播到集群的所有节点上，而只更新当前的数据节点和备份的数据节点，如图 7 所示。

当读取数据的时候，我们就不能像 replicated Cache 那样，直接将 Cache 从集群的任一节点上取到，因为集群的数据是分布在每个节点上的。这时我们可能要经过一次网络传输，集群的另一节点读到这个数据。整个数据读取的情形如图 8 所示。

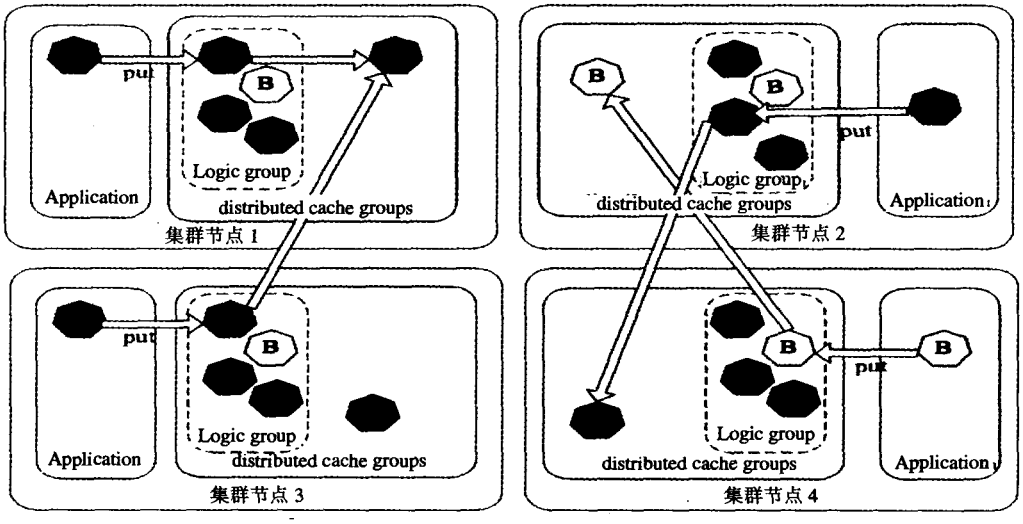

图 8　Distributed Cache 数据获取的过程

经过分析，Distributed Cache 具有以下特点：1）Distributed Cache 对于经常更新的数据性能要比 Replicated Cache 好；2）由于 Distributed Cache 将节点数据分布在 cluster 的不同节点上，因此它可以更有效地利用内存资源；3）由于 Distributed Cache 的数据分布策略远比 Replicated Cache 复杂，因此它的实现算法也相当复杂。它也使数据的复杂负载均衡及失败恢复算法也变得很复杂。

5　Cache 数据的负载均衡

5.1　负载均衡的基本思想

负载均衡是指使集群上的每个节点保持基本相同的负载。如果某个节点负载过重，则它会将部分数据的负载转移给负载较轻的节点。这样，就可以保证整个集群的可伸缩性。一般的负载均衡器都是用硬件来实现的，会自动判断将下一个 request 提交给哪个集群节点。

对于 Replicated Cache 来说，每个节点的数据是一样的，亦即每个节点是等价的。因此我们不需要做太多的负载均衡工作，负载均衡器会自动将负载分配给负载轻的节点。因此，我们在这里只考虑 Distributed Cache 的负载均衡算法。

我们来考虑以下几种情况的负载均衡：

1）新节点加入 Cluster。由于新节点上没有数据，因此它的负载是最轻的，其它节点的负载会分配到这个节点上。这里需要简单的算法，实现新加入的节点负载经过调节以后与其它节点一样。

2）节点退出 Cluster。当某个节点要退出 cluster 时，它所负载的那部分数据会分配给其它节点。这个过程与新节点加入刚好相反，但最终的结果仍然是希望经过调整使每个 cluster 节点的访问量一样。

3）某个节点负载过重。由于每个节点上访问的 Cache 数据量是基本一致的，因此出现某个节点负载量过重的原因是节点内的某些数据比其它数据更频繁地访问。这时，我们在调整时会根据访问数将访问量最大的节点数据与负载轻的节点上的访问量小的数据进行交换，最终达到负载的平衡。

5.2　负载均衡的数据跟踪

当数据在节点上分布时，每个节点的数据都是不同的，而且由于负载均衡与失败恢复所涉及到的算法相当复杂，我们必须有一种机制跟踪每个节点的状况。

首先我们假设 cluster 上有 k 个节点，在某一时刻 Cache 共有 n 条数据，则我们需要将这些数据分成 k 份，理想情况下每个节点上数据数量应该是 n/k 或 $n/k+1$。但是现实中由于数据的批处理等问题，可能会加上一些误差，设为 z，则每个节点的数据量应该是 $n/k+-z$。为了跟踪这些数据，我们需要将这些已分好的每部分数据都取一个名称，设为 A、B、C …（为确保名称的唯一性，在实现里取的名字会比这个复杂），则我们可以通过建立一个表格来跟踪数据变化，其示例如下表 1。

表 1

节点编号	数据名称	数据量	备份数据名称	备份数据量
1	A	n/k	B	$n/k-1$
2	B	$n/k-1$	C	n/k
3	C	n/k	D	$n/k+1$
4	D	$n/k+1$	A	n/k

现在我们来考察以下几种情况下发生的数据迁移及变化：

1）有 a 条数据插入到 Cache 中。

理想情况下，我们可以将这批数据均匀地分配到各个节点。它的最终结果应该如表 2。

表 2

节点编号	数据名称	数据量	备份数据名称	备份数据量
1	A	$(n+a)/k$	B	$(n+a)/k-1$
2	B	$(n+a)/k-1$	C	$(n+a)/k$
3	C	$(n+a)/k$	D	$(n+a)/k+1$
4	D	$(n+a)/k+1$	A	$(n+a)/k$

2)有 a 条数据被删除。

最终的节点分布情况与第 1 种情况类似,结果如表 3。

表 3

节点编号	数据名称	数据量	备份数据名称	备份数据量
1	A	$(n-a)/k$	B	$(n-a)/k-1$
2	B	$(n-a)/k-1$	C	$(n-a)/k$
3	C	$(n-a)/k$	D	$(n-a)/k+1$
4	D	$(n-a)/k+1$	A	$(n-a)/k$

3)新加入一个 cluster 节点。

节点数增加后,每个节点所负载的数据量都减轻了。我们通过一个后台线程将一部分数据转移到新的节点,并可对这个节点的数据给予新的名称并命名,如表 4 所示。

表 4

节点编号	数据名称	数据量	备份数据名称	备份数据量
1	A	$n/(k+1)$	B	$n/(k+1)-1$
2	B	$n/(k+1)-1$	C	$n/(k+1)$
3	C	$n/(k+1)$	D	$n/(k+1)+1$
4	D	$n/(k+1)+1$	E	$n/(k+1)$
5	E	$n/(k+1)$	A	$n/(k+1)$

注意备份数据域的转移,我们这里将 D 的备份数据换成了 E,并将 E 的备份数据换成了 A,这就保证了 cluster 上每个节点都有不同的备份数据,同时也使 cluster 上每个节点的数据都是不同的。

6 实验与结果分析

6.1 实验环境

我们的实验环境里配置了 3 台电脑组成一个集群,并用另一台电脑作为负载均衡器。客户端采用 Apache JMeter[9] 作为性能测试的工具。测试的系统是我们实验室开发的中医药公共卫生辅助决策平台,后台数据库的总数据库量达 20 万条记录。

具体的实验环境如图 9 所示。

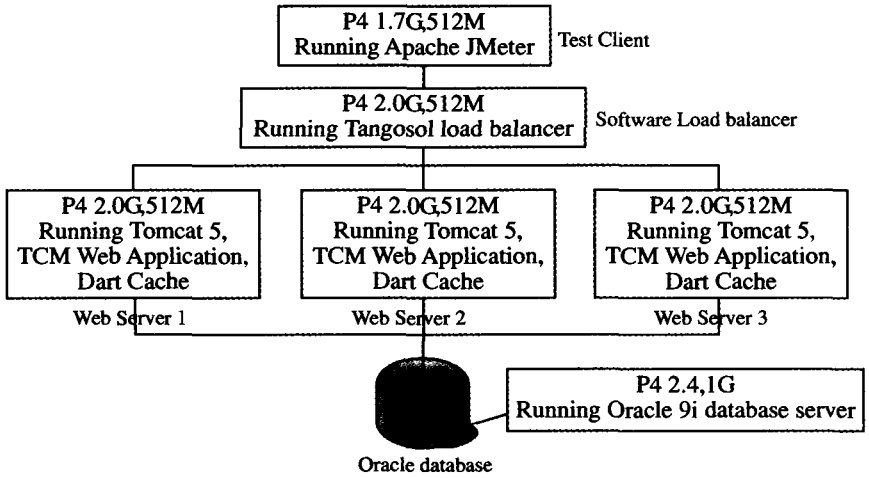

图 9 DartCache 性能测试环境

6.2 实验结果

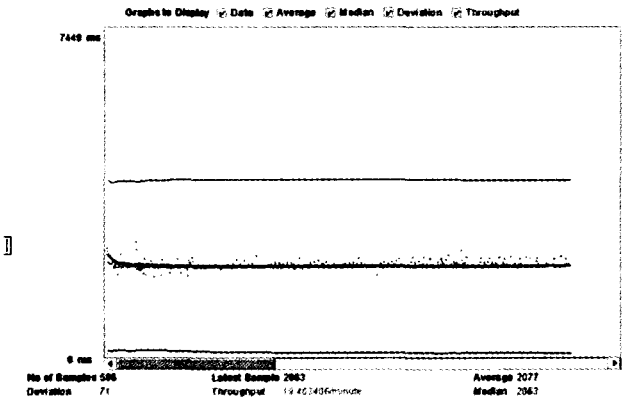

图 10 单线程不使用 Cache 的性能测试结果

我们的实验结果分成两组,一组只产生 1 个用户请求,这组实验完全用于测试使用 DartCache 后对系统的性能影响;另一组则产生 20 个用户请示,这组实验用于测试系统使用 DartCache 后对系统的性能和可伸缩性的影响。

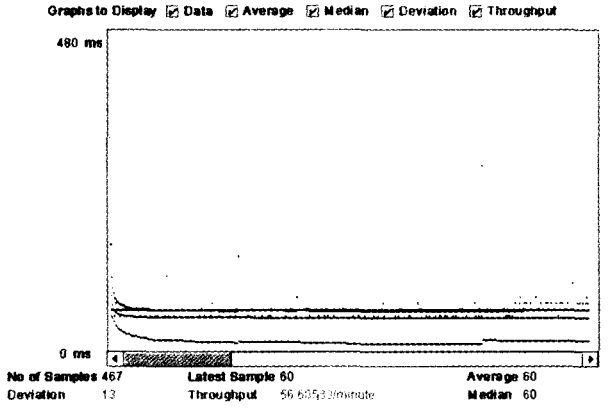

图 11 单用户使用 Cache 的性能测试结果

如图 10 和 11 所示的第 1 组实验中,在没有 Cache 的情

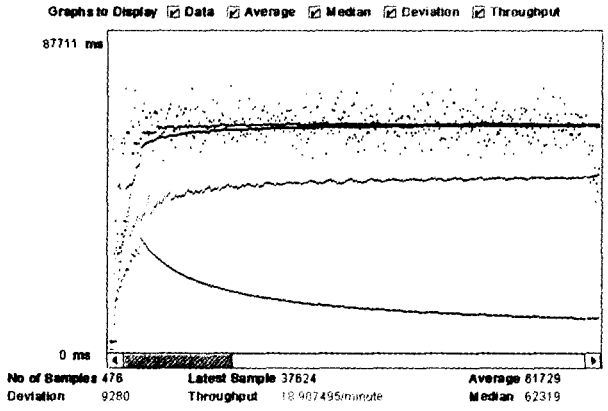

图 12 20 个线程不使用 Cache 的性能测试结果

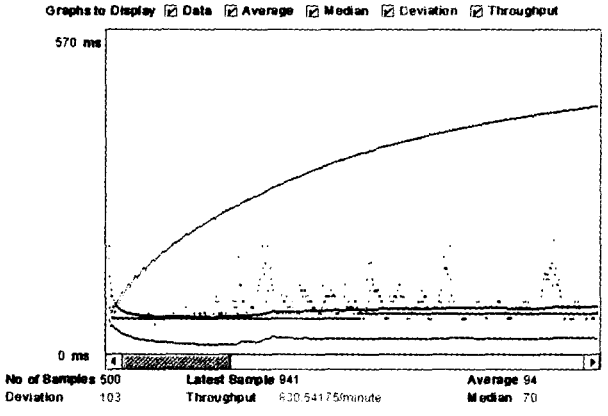

图 13 20 个线程使用 Cache 的性能测试结果

况下，系统的平均响应时间达 2077ms。使用 Cache 以后，系统的平均响应时间缩短为 60ms。由于第 1 组实验的用户 thread 只有 1 个，因此系统的吞吐量未达到满负荷，不能作为参考依据。

如图 12 和 13，在第 2 组实验中，当用户数达 20 个后，没有使用 DartCache 时系统的平均响应时间为 61729ms，使用 DartCache 后为 94ms。吞吐量也由 18.9 请求/分变成 830.5 请求/分。

实验证明，在使用了 DartCache 后，系统的性能和可能伸缩性都得到了巨大的提高。

结束语 采用 Cache 技术来提高数据库的访问性能是一种很常用的技术，已在工业界广泛使用。但是在 Cache 数据的分布性与 AOP 的设计上仍然有很多地方可以提高。

我们设计的 DartCache 是完整的分布式 Cache 解决方案，它在 Cache 数据分布与 AOP 设计上比以往的解决方案有了很大的改进。最后的实验结果证明，使用 DartCache 可使系统性能大幅度提高。

参 考 文 献

1 Fowler M. Patterns of Enterprise Application Architecture. Addison Wesley, 2002
2 Deutsch P. 分布式计算八大谬论. http://today.java.net/jag/Fallacies.html. 2002
3 Neward T. Effective Enterprise Java. Addison Wesley, 2004
4 EhCache. http://ehcache.sourceforge.net/. 2004
5 OpenSymphony. OSCache. http://www.opensymphony.com. 2004
6 JBoss. JBoss Cache. http://www.jbosss.org
7 Tangosol. Tangosol Coherenece. http://www.tangosol.com
8 Ban B. Design and Implementation of a Reliable Group Communication Toolkit for Java. Cornell University, 1999
9 Apache. JMeter. http://jakarta.apache.org/jmeter/

(上接第 150 页)

包括发送到目的端以及目的端返回。

(4)对于算法 report-every-l：

report-every-l 算法是 collect-en-route 算法和 report-en-route 算法的折中。collect-en-route 算法具有线型的消息复杂度和二次项的延迟；而 report-en-route 算法具有线型的延迟和二次项的消息复杂度。综合这两种算法可以定义 report-every-l 算法，使消息、时间复杂度都比较理想，该算法优化了前面两种算法。算法 report-every-l 思想是将 n 分成 n/l 段，每段长度为 l，为所有 n/l 段发送一个固定大小的初始化为 collect-en-route 算法的消息(可以在每段开始位置初始化一个赋值为 l 的计数器，每通过一个节点减一来实现)。所以对第 i 段来说其至少要经历 $(i-1)(C+P)$ 时间单元后才开始执行 collect-en-route，所以有：

$$TC(n)+(n-l)(C+P)+\sum_{i=1}^{l}(C+iP)=O(nC+(n+l^2)P)$$

$$MC(n)=O(n)+\sum_{i=1}^{n/l}(l+il)=O(\frac{n^2}{l})$$

如果选择 $l=\sqrt{n}$，就可以使得 $TC(\sqrt{n})$ 具有线性复杂度，而消息复杂度为 $O(n\sqrt{n})$。为了平衡两种复杂度，令 $l^2=n^2/l$，则两种复杂度都为 $O(n^{4/3})$。

综上，对本文的几种常用路径转发算法总结如下：

Algorithms	TC	MC
get-response	$O(np+n^2C)$	$O(n^2)$
report-en-rout	$O(np+nC)$	$O(n^2)$
collect-en-route	$O(n^2p+nC)$	$O(n)$
report-every-l	$O((n+l^2)p+nC)$	$O(n)$

结束语 基于主动网络技术的网络管理是当今网络管理界研究的一个热点，本文讨论了基于主动节点的网管模型，对模型中的管理报文的结构、转发机制、转发算法进行了分析。主动网络是一种运行时可编程、可扩展的网络，提供了一种动态的运行环境。当网管报文到达网管主动节点时，网管主动节点将数据和程序代码分离，在执行环境中执行这些代码。网管报文由传统的 UDP 报头、ANEP 报头、主动报文主体和有效负载组成，报文主体遵循 ANTS 封装体的构造方式。在设计网络管报文中要充分考虑和利用主动网络的特点，采用恰当的包转发模型和算法。将具有网络管理功能的报文动态地分布在主动节点上，利用主动节点的计算能力，使节点能够自动发现、及时处理问题，是本系统实现网络优化管理的基本思想。把分布式计算模型引入到网络体系结构和网管体系结构是网管发展趋势，也是从根本上解决问题的方法。

参 考 文 献

1 Fatta G D, Gaglio S, Re G L, Ortolani M[J]. Adaptive Routing in Active Networks. IEEE Openarch 2000, Tel Aviv Israel 23-24 March 2000
2 Fatta G D, Re G L. Active Networks[J]. an Evolution of the Internet. In: Proc. of AICA2001 - 39th Annual Conference, Cernobbio, Italy, Sept. 2001
3 Munir S. Active Networks: A Survey[EB/OL]. http://www.cse.ohio-state.edu/jain/cis788-97/ ftp/ activenets/index.htm, 2000-07-02
4 Shaer E A. Active Management Framework for Distributed Multimedia Systems[J]. Journal of Networks and Systems Management, 2000, 8(1): 49～72
5 Kiwior D, Zabele S. Active Resource allocation in Active Networks [J]. IEEE JSAC, 2000, 19(3): 452～459
6 Brunner M, Stadler R. Service Management in Multi-Party Active Networks[J]. IEEE Communications Magazine, 2000, 38(3): 281～286
7 Calvert K L. Directions in Active Networks [J]. IEEE Communications Magazine, 1998, 36 (1): 72～78
8 Kawamura R, Stadler R. Active Distributed Management for IP Networks [J]. IEEE Communications Magazine, 2000, 38(4): 114～121