Amoeba:分布式数据库 Proxy 解决方案

author：陈思儒

随着传统的数据库技术日趋成熟、计算机网络技术的飞速发展和应用范围的扩充，数据库应用已经普遍建立于计算机网络之上。这时集中式数据库系统表现出它的不足：集中式处理，势必造成性能瓶颈；应用程序集中在一台计算机上运行，一旦该计算机发生故障，则整个系统受到影响，可靠性不高；集中式处理引起系统的规模和配置都不够灵活，系统的可扩充性差。在这种形势下，集中式数据库将向分布式数据库发展。

分布式数据库系统的优点：

1、降低费用。分布式数据库在地理上可以式分布的。其系统的结构符合这种分布的要求。允许用户在自己的本地录用、查询、维护等操作，实行局部控制，降低通信代价，避免集中式需要更高要求的硬件设备。而且分布式数据库在单台机器上面数据量较少，其响应速度明显提升。

2、提高系统整体可用性。避免了因为单台数据库的故障而造成全部瘫痪的后果。

3、易于扩展处理能力和系统规模。分布式数据库系统的结构可以很容易地扩展系统，在分布式数据库中增加一个新的节点，不影响现有系统的正常运行。这种方式比扩大集中式系统要灵活经济。在集中式系统中扩大系统和系统升级，由于有硬件不兼容和软件改变困难等缺点，升级的代价常常是昂贵和不可行的。

Amoeba 在分布式数据库系统充当什么角色？

Amoeba 在分布式数据库领域将致力解决数据切分，应付客户端“集中式”处理分布式数据。这儿集中式是一个相对概念，客户端不需要知道某种数据的物理存储地。避免这种逻辑出现在业务端，大大简化了客户端操作分布式数据的复杂程度。

Amoeba Overview

Amoeba 属于分布式数据库代理开发框架，目前基于 amoeba 的数据库分布式代理有 Amoeba For Mysql。

以下链接提供了您在开发、使用、安装 Amoeba 时，可能需要的信息：

- Amoeba 是什么

本节描述了 Amoeba 是什么的以及它的框架

- **Amoeba For Mysql**

本节介绍如何快速配置 Amoeba For Mysql，如何启动 Amoeba For Mysql，和您必须确保已经具备所有先决条件

- **Amoeba 高级使用**

本节主要讲述如何利用 Amoeba 为目标数据库服务器 创建 负责均衡、数据切分、读写分离以及 Amoeba 本身性能调优方面

- **如何在 Amoeba 上面进行高级开发**

为其他 数据库 开发 Amoeba 新实例、数据库相关的函数开发等。

Amoeba 是什么

Amoeba(变形虫)项目，专注 分布式数据库 proxy 开发。座落与 Client、DB Server(s)之间。对客户端透明。具有负载均衡、高可用性、sql 过滤、读写分离、可路由相关的 query 到目标数据库、可并发请求多台数据库合并结果。

主要解决：

- 降低 数据切分带来的复杂多数据库结构
- 提供切分规则并降低 数据切分规则 给应用带来的影响
- 降低 db 与客户端的连接数
- 读写分离

Amoeba Architecture

Amoeba 作为 DataBase Proxy 的开发框架。致力于解决数据切分、读写分离。以下将为您介绍 Amoeba 框架

- **Built on Java NIO**

  1. 采用 java NIO 框架无阻塞模式，不像传统的 Socket 编程在大量并发的情况非常浪费系统资源、而且可扩展性也较差

- **Reusable Server Connection**

  Amoeba 提供与数据库连接的可重用度非常高，在 Amoeba 系统内所 Database Connection 同时共享给所有连接到 Amoeba 的客户端

- **提供读写分离、数据切分**

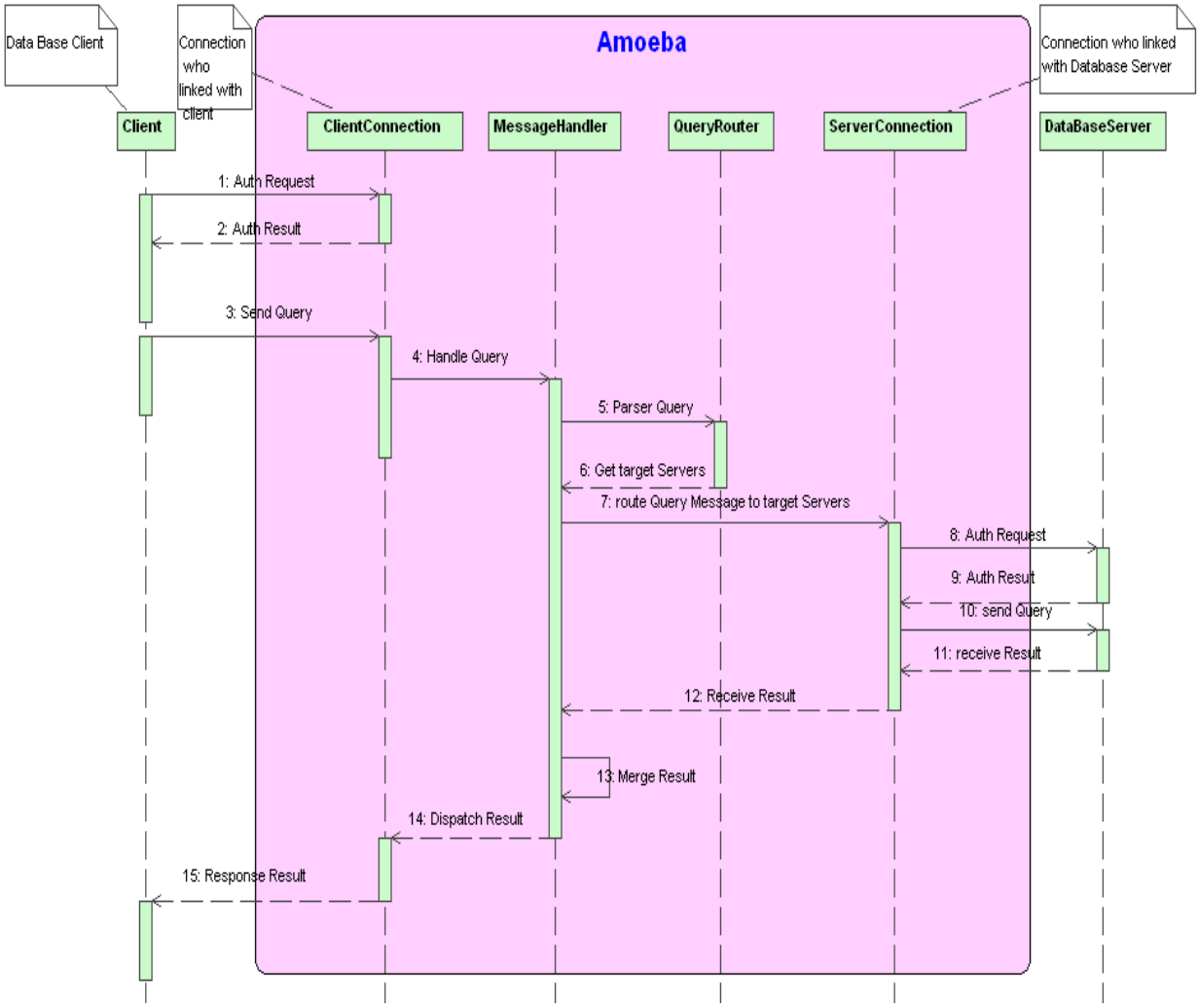
  1. 传统的读写分离技术需要通过客户端或者相关的 Database Driver 技术才能解决，而且客户端的配置也比较复杂

  2. 单台 Database 性能总是有限制的，基于 Amoeba 上面可以寻找一种可线性扩展的多数据库支持。Amoeba 为 DBA 提供一种非常友好的类似 SQL 语法的数据切分规则同时客户端不用担心过多的 DataBase Server 会给应用带来更多的配置。

- **支持高可用性、负责均衡**

1. Amoeba 提供 Database 连接的异常检测与连接恢复功能。
2. 用户可节省使用其他昂贵的负载均衡的硬件设备，Amoeba 提供多台 Database Server 负载均衡策略（轮询、当前活动连接数量）

Amoeba Sequence

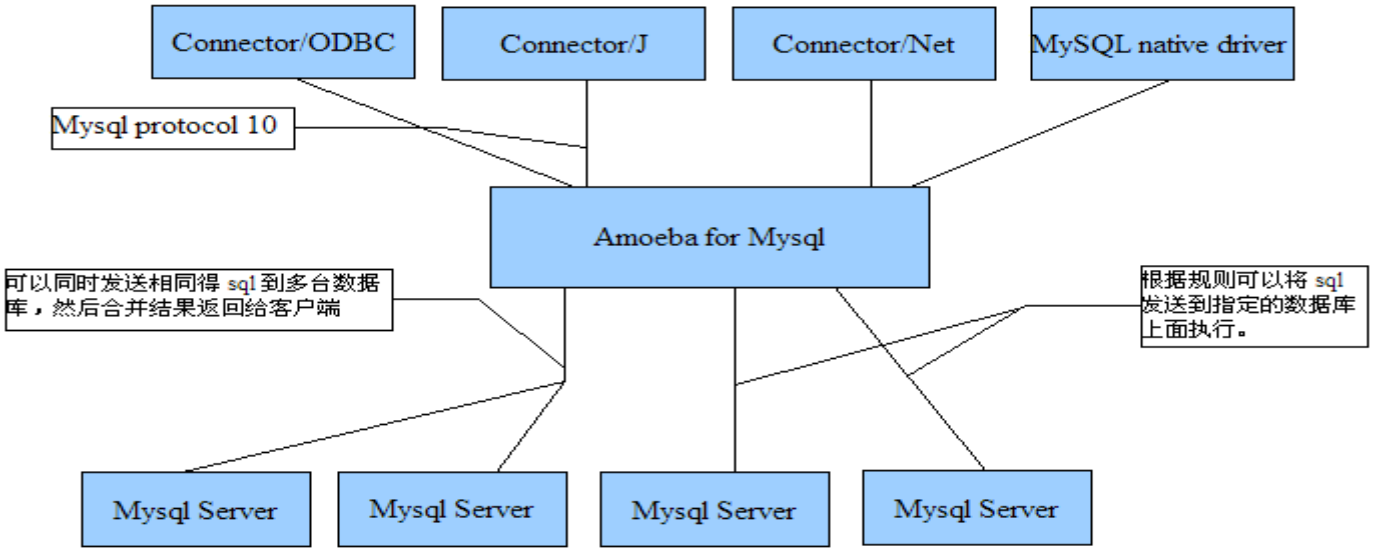


Amoeba For Mysql

Amoeba For Mysql 是 Amoeba 项目的子项目。要使用 Amoeba For Mysql 您必须确保您已符合所有先决条件：

先决条件：

1. Java SE 1.5 或以上 Amoeba 框架是基于 JDK1.5 开发的，采用了 JDK1.5 的特性。
2. 支持 Mysql 协议版本 10（mysql 4.1 以后的版本）。
3. 您的网络环境至少运行有一个 mysql 4.1 以上的服务



如何快速配置：

1. **配置 Server(以下是双核 CPU 配置，调整线程数可优化性能),配置说明：**

配置项	是否必选	默认值	说明
	否	8066	Amoeba Server 绑定的对外端口
ipAddress	否	空	Amoeba 绑定的 IP
user	是	空	客户端连接到 Amoeba 的用户名
password	否	空	客户端连接到 Amoeba 所用的密码
readThreadPoolSize	否	16	负责读客户端、databa seserver 端网络数据包线程数
clientSideThreadPoolSize	否	16	负责读执行客户端请求的线程数
serverSideThreadPoolSize	否	16	负责处理服务端返回数据包的线程数

Server Tag Configuration

```
<server>
    <!-- proxy server绑定的端口 -->
    <property name="port">2066</property>

    <!-- proxy server绑定的IP -->
    <property name="ipAddress">127.0.0.1</property>

    <!-- proxy server net IO Read thread size -->
    <property name="readThreadPoolSize">100</property>

    <!-- proxy server client process thread size -->
    <property name="clientSideThreadPoolSize">80</property>

    <!-- mysql server data packet process thread size -->
    <property name="serverSideThreadPoolSize">100</property>

    <!-- 对外验证的用户名 -->
    <property name="user">root</property>

    <!-- 对外验证的密码 -->
    <property name="password">password</property>
</server>
```

2. **配置 ConnectionManager**

需要至少配置一个 ConnectionManager，每个 ConnectionManager 将作为一个线程启动，ConnectionManager 负责管理所注册在自身的 Conneciton、负责他们的空闲检测，死亡检测、IO Event

```
<!--
    每个 ConnectionManager 都将作为一个线程启动。
    manager 负责 Connection IO 读写/死亡检测
-->
<connectionManagerList>
    <connectionManager name="defaultManager">

<className>com.meidusa.amoeba.net.AuthingableConnectionManager</className>
    </connectionManager>
    <connectionManager name="default"/>
</connectionManagerList>
```

3. **配置 dbServer，需要至少配置一个 dbServer，每个 dbServer 将是物理数据库 Server 的衍射** factoryConfig --目标物理数据库衍射配置情况：

配置项	是否必选	默认值	说明
manager	是	空	表示该 dbServer 将注册到指定的 ConnectionManager
port	否	3306	目标数据库端口
ipAddress	否	127.0.0.1	目标数据库 IP
schema	否	空	连接初始化的 Schema

配置项	是否必选	默认值	说明
user	是	空	用于登陆目标数据库的用户名
password	否	空	用于登陆目标数据库的密码
className	是	空	连接工厂实现类(com.meidusa.amoeba.mysql.net.MysqlServerConnectionFactory)

4. poolConfig -- 连接池配置情况：

配置项	是否必选	默认值	说明
className	否		连接池实现类。默认：com.meidusa.amoeba.net.poolable.PoolableObjectPool
maxActive	否	8	最大活动连接数，如果达到最大活动连接数，则会等待
maxIdle	否	8	最大的空闲连接数，如果超过则将会关闭多余的空闲连接
minIdle	否	0	最小的空闲连接，连接池将保持最小空闲连接，即使这些连接长久不用
testOnBorrow	否	false	当连接在使用前是否检查连接可用
testWhileIdle	否	fale	是否检测空闲连接，这个参数启动的时候下列 2 个参数才有效
minEvictableIdleTimeMillis	否	30 分钟	连接空闲多少时间将被驱逐（关闭）(time Unit:ms)
timeBetweenEvictionRunsMillis	否	-1	用于驱逐空闲连接没间隔多少时间检查一次空闲连接(time Unit:ms)

dbServer Tag Configuration

```
<dbServerList>
        <!--
            一台 mysqlServer 需要配置一个 pool，
            如果多台 平等的 mysql 需要进行 loadBalance，
            平台已经提供一个具有负载均衡能力的 objectPool：com.meidusa.amoeba.mysql.server.MultipleServerPool
            简单的配置是属性加上 virtual="true",该 Pool 不允许配置 factoryConfig
            或者自己写一个 ObjectPool。
        -->
        <dbServer name="userdb">

            <!-- PoolableObjectFactory 实现类 -->
            <factoryConfig>
                <className>com.meidusa.amoeba.mysql.net.MysqlServerConnectionFactory</className>
                <property name="manager">defaultManager</property>

                <!-- 真实 mysql 数据库端口 -->
                <property name="port">3301</property>

                <!-- 真实 mysql 数据库 IP -->
                <property name="ipAddress">127.0.0.1</property>

                <!-- 用于登陆 mysql 的用户名 -->
                <property name="user">root</property>
                <property name="password">password</property>

                <property name="schema">test</property>
            </factoryConfig>
```

```
        <!-- ObjectPool 实现类  -->
        <poolConfig>
            <className>com.meidusa.amoeba.net.poolable.PoolableObjectPool</className>
            <property name="maxActive">200</property>
            <property name="maxIdle">200</property>
            <property name="minIdle">10</property>
            <property name="minEvictableIdleTimeMillis">600000</property>
            <property name="timeBetweenEvictionRunsMillis">600000</property>
            <property name="testOnBorrow">true</property>
            <property name="testWhileIdle">true</property>
        </poolConfig>
    </dbServer>

    <dbServer name="blogdb" >

        <!-- PoolableObjectFactory 实现类  -->
        <factoryConfig>
            <className>com.meidusa.amoeba.mysql.net.MysqlServerConnectionFactory</className>
            <property name="manager">defaultManager</property>
            <property name="port">3302</property>
            <property name="ipAddress">127.0.0.1</property>
            <property name="user">blog</property>
            <property name="password">blog</property>
            <property name="schema">test</property>
        </factoryConfig>

        <!-- ObjectPool 实现类  -->
        <poolConfig>
            <!--
            <className>com.meidusa.amoeba.net.poolable.PoolableObjectPool</className>
            -->
            <property name="maxActive">200</property>
            <property name="maxIdle">200</property>
            <property name="minIdle">10</property>
            <property name="minEvictableIdleTimeMillis">600000</property>
            <property name="timeBetweenEvictionRunsMillis">600000</property>
            <property name="testOnBorrow">true</property>
            <property name="testWhileIdle">true</property>
        </poolConfig>
    </dbServer>
    <dbServer name="other" >

        <!-- PoolableObjectFactory 实现类  -->
        <factoryConfig>
            <className>com.meidusa.amoeba.mysql.net.MysqlServerConnectionFactory</className>
            <property name="manager">defaultManager</property>
            <property name="port">3303</property>
            <property name="ipAddress">127.0.0.1</property>
            <property name="user">root</property>
            <property name="password">password</property>
            <property name="schema">test</property>
        </factoryConfig>

        <!-- ObjectPool 实现类  -->
        <poolConfig>
            <!--
            <className>com.meidusa.amoeba.net.poolable.PoolableObjectPool</className>
            -->
            <property name="maxActive">200</property>
            <property name="maxIdle">200</property>
            <property name="minIdle">10</property>
            <property name="minEvictableIdleTimeMillis">600000</property>
            <property name="timeBetweenEvictionRunsMillis">600000</property>
            <property name="testOnBorrow">true</property>
            <property name="testWhileIdle">true</property>
```

```
                </poolConfig>
            </dbServer>

        <dbServer name="virtualdb" virtual="true">
                <poolConfig>
                    <className>com.meidusa.amoeba.server.MultipleServerPool</className>
                    <!-- 负载均衡参数  1=ROUNDROBIN , 2=WEIGHTBASED -->
                    <property name="loadbalance">1</property>

                    <!-- 参与该 pool 负载均衡的 poolName 列表以逗号分割,server1、server2、server3 这几个 dbserver 配置
省略，他们几个是平等的数据库     -->
                    <property name="poolNames">server1,server2,server3</property>
                </poolConfig>
            </dbServer>
</dbServerList>
```

5. QueryRouter 查询路由配置

配置项	是否必选	默认值	说明
className	是	空	QueryRouter 实现类，Amoeba For Mysql（com.meidusa.amoeba.mysql.parser.MysqlQueryRouter）。
functionConfig	否	空	用于解析 sql 函数的配置文件，如果不配置则将不解析包含函数 sql 或者解析的不完整。
ruleConfig	否	空	数据切分规则配置文件，如果不配置则 sql 数据切分功能将不能用
needParse	否	true	是否对 sql 进行 parse，如果 false 则将不能使用数据切分、读写分离等功能
defaultPool	是	空	needParse=false、无法解析 query、不满足切分规则的、writePool|readPool == null 情况。 所有 sql 将在默认得 dbServer 上面执行。(必选)
writePool	否	空	启用 needParse 功能，并且没有匹配到数据切分规则，则 update、insert、delete 语句将在这个 pool 中执行
readPool	否	空	启用 needParse 功能，并且没有匹配到数据切分规则，则 select 语句将在这个 pool 中执行
LRUMapSize	否	1000	statment cache ，存放 sql 解析后得到的 statment

Amoeba 高级使用指南

- 基于 Amoeba 的读写分离

  在不影响应用的情况下，引入 amoeba 将透明地解决数据读写分离的技术方案。

- 基于 Amoeba 的数据垂直切分

  垂直切分（纵向）数据是数据按照网站业务、产品进行切分，比如用户数据、博客文章数据、照片数据、标签数据、群组数据等等每个业务一个独立的数据库或者数据库服务器。引入 Amoeba 将不需要客户端更多的干预就能很好地定位这些数据。

- 基于 Amoeba 数据水平切分

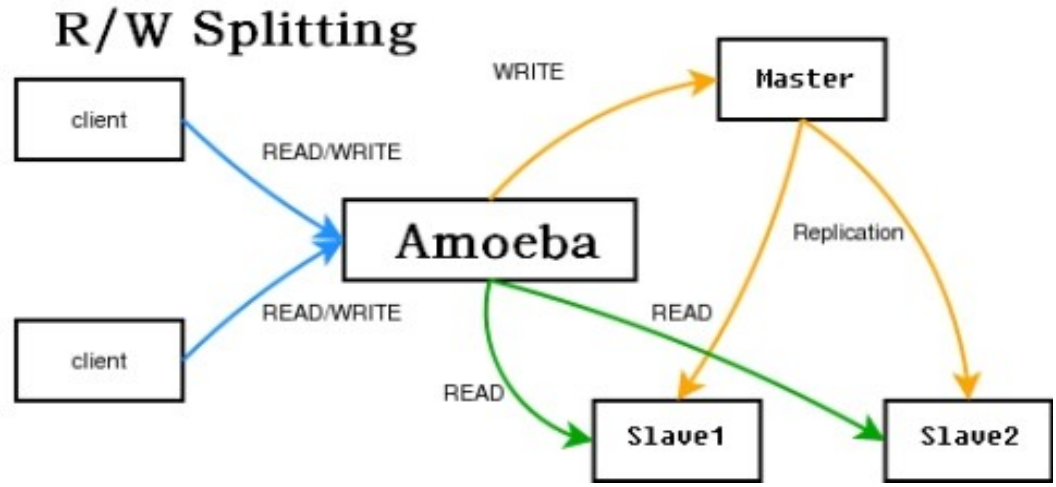
  应付绝大多数的切分规则，轻易解决数据水平切分。让你的数据库系统更具有扩展性能。更具承载能力。

Master/Slave 结构之下的读写分离

首先说明一下 amoeba 跟 mysql proxy 在读写分离的使用上面的区别。

在 mysql proxy 6.0 版本 上面如果想要读写分离并且 读集群、写集群 机器比较多情况下，用 mysql proxy 需要相当大的工作量，目前 mysql proxy 没有现成的 lua 脚本。mysql proxy 根本没有配置文件，lua 脚本就是它的全部，当然 lua 是相当方便的。那么同样这种东西需要编写大量的脚本才能完成一个复杂的配置。而 amoeba 只需要进行相关的配置就可以满足需求。

示意图：



一、Master/Slave 结构读写分离：

Master： master （可读写）

slaves：slave1、slave2、(2 个平等的数据库。只读/负载均衡）

amoeba 提供读写分离 pool 相关配置。并且提供负载均衡配置。

可配置 slave1、slave2 形成一个虚拟的 virtualSlave,该配置提供负载均衡、failOver、故障恢复功能

```
<dbServer name="virtualSlave" virtual="true">
  <poolConfig>
    <className>com.meidusa.amoeba.server.MultipleServerPool</className>
    <!-- 负载均衡参数  1=ROUNDROBIN , 2=WEIGHTBASED -->
    <property name="loadbalance">1</property>

    <!-- 参与该 pool 负载均衡的 poolName 列表以逗号分割  -->
    <property name="poolNames">slave1,slave2</property>
  </poolConfig>
</dbServer>
```

如果不启用数据切分，那么只需要配置 QueryRouter 属性

wirtePool= master

readPool=virtualSlave

```
<queryRouter>
  <className>com.meidusa.amoeba.mysql.parser.MysqlQueryRouter</className>
  <property name="LRUMapSize">1500</property>
  <property name="defaultPool">master</property>

  <property name="writePool">master</property>
  <property name="readPool">virtualSlave</property>

  <property name="needParse">true</property>
</queryRouter>
```

那么遇到 update/insert/delete 将 query 语句发送到 wirtePool，将 select 发送到 readPool 机器中执行。

基于Amoeba 数据垂直切分

垂直切分（纵向）数据是数据按照网站业务、产品进行切分，比如用户数据、博客文章数据、照片数据、标签数据、群组数据等等每个业务一个独立的数据库或者数据库服务器。

如果一个应用只针对单纯的业务功能模块。那么可以直接连接相应的被垂直切分的数据库。但一些复杂的应用需要用到相当多的业务数据，涉及到几乎所有的业务数据。那么垂直切分将会给应用带来一定的复杂度，而且对工程师开发也会有一定影响。整个应用的复杂度将上升。

Amoeba 在其中充当了门面功能，相当于水利枢纽。疏通应用与多个数据库数据通讯。

如何垂直切分功能：

假设有 3 个数据库：userdb、blogdb、otherdb

userdb：包含有 user 表

blogdb：包含有 message、event、

otherdb：其他表所在的数据库

1、在 amoeba 安装目录下面 config/amoeba.xml 配置文件中，启用 ruleConfig 配置。

这儿省略了 dbServer 的配置，相关配置请参阅 dbServer 配置

```
<queryRouter>
        <className>com.meidusa.amoeba.mysql.parser.MysqlQueryRouter</className>
        <property name="ruleConfig">${amoeba.home}/conf/rule.xml</property>
        <property name="functionConfig">${amoeba.home}/conf/functionMap.xml</property>
        <property name="LRUMapSize">1500</property>
        <property name="defaultPool">otherdb</property>
        <property name="needParse">true</property>
</queryRouter>
```

2、rule.xml 配置文件相应配置,假设 schema 均为 test

```
<?xml version="1.0" encoding="gbk"?>
<!DOCTYPE amoeba:rule SYSTEM "rule.dtd">
<amoeba:rule xmlns:amoeba="http://amoeba.meidusa.com/">
    <tableRule name="USER" schema="test" defaultPools="userdb"/>
    <tableRule name="MESSAGE" schema="test" defaultPools="blogdb"/>
     <tableRule name="EVENT" schema="test" defaultPools="blogdb"/>
</amoeba:rule>
```

接下来就可以看到相应的效果：

select * from user where user_name = 'myname'; 将会到 userdb 相应的数据库上面执行。

Insert into message(id,gmt_create_time,msg) values(111,now(),'only for test');将会在 blogdb 上面执行

简单的配置即可完成基于 amoeba 的数据垂直切分方案。

3、amoeba 不仅仅可以读写分离、垂直切分，而且可以结合 2 者进行在垂直切分上面再进行读写分离。以 userdb 举例：构建 master/slave 结构

master-userdb:可读写的 mysql master

virtual-userdb:可读的虚拟的 slave（virtual db server 配置请参阅 virtual db 配置）

```
<?xml version="1.0" encoding="gbk"?>
<!DOCTYPE amoeba:rule SYSTEM "rule.dtd">
<amoeba:rule xmlns:amoeba="http://amoeba.meidusa.com/">
    <tableRule name="USER" schema="test" defaultPools="master-userdb"
readPools="virtual-userdb" writePools="master-userdb"/>
    <tableRule name="MESSAGE" schema="test" defaultPools="blogdb"/>
     <tableRule name="EVENT" schema="test" defaultPools="blogdb"/>
</amoeba:rule>
```

基于 Amoeba 数据水平切分

引用“可伸缩性最佳实践：来自 eBay 的经验”：

——

按功能分割对我们的帮助很大，但单凭它还不足以得到完全可伸缩的架构。即使将功能一一解耦，单项功能的资源需求随着时间增长，仍然有可能超出单一系 统的能力。我们常常提醒自己，“没有分割就没有伸缩”。在单项功能内部，我们需要能把工作负载分解成许多我们有能力驾驭的小单元，让每个单元都能维持良好 的性能价格比。这就是水平分割出场的时候了。

在应用层次，由于 eBay 将各种交互都设计成无状态的，所以水平分割是轻而易举之事。用标准的负载均衡服务器来路由进入的流量。所有应用服务器都是 均等的，而且任何服务器都不会维持事务性的状态，因此负载均衡可以任意选择应用服务器。如果需要更多处理能力，只需要简单地增加新的应用服务器。

数据库层次的问题比较有挑战性，原因是数据天生就是有状态的。我们会按照主要的访问路径对数据作水平分割（或称为“sharding”）。例如用户 数据目前被分割到 20 台主机上，每台主机存放 1/20 的用户。随着用户数量的增长，以及每个用户的数据量增长，我们会增加更多的主机，将用户分散到更多的 机器上去。商品数据、购买数据、帐户数据等等也都用同样的方式处理。用例不同，我们分割数据的方案也不同：有些是对主键简单取模（ID 尾数为 1 的放到第一 台主机，尾数为二的放到下一台，以此类推），有些是按照 ID 的区间分割（1-1M、1-2M 等等），有些用一个查找表，还有些是综合以上的策略。不过具体 的分割方案如何，总的思想是支持数据分割及重分割的基础设施在可伸缩性上远比不支持的优越。

——

ebay 的数据水平切分提案应该是大家都能想到的，但数据切分以后我们如何访问我们的应用，我们应用如何按照规则做实时的数据切分？在应用层面还是其他层？这个难题可以托付给 amoeba 来解决。Amoeba 提供对 dba 非常友好的数据切分规则表达式。

假设我们 messagedb 需要根据 id hash 进行水平切分，我们可以根据 hash 范围分成 2 台：

规则 1： abs(hash(id)) mod 2 = 0 → blogdb-1

规则 2： abs(hash(id)) mod 2 =1 →blogdb-2

这儿 abs、hash 都是 amoeba 规则中使用到的函数，amoeba 允许开发人员增加新的规则函数。

具体规则配置(下面 blogdb-1、blogdb-2 配置请参阅 dbserver 配置):

```
<?xml version="1.0" encoding="gbk"?>
<!DOCTYPE amoeba:rule SYSTEM "rule.dtd">
<amoeba:rule xmlns:amoeba="http://amoeba.meidusa.com/">
    <tableRule name="USER" schema="test" defaultPools="master-userdb"
readPools=”virtual-userdb” writePools=”master-userdb”/>
    <tableRule name="MESSAGE" schema="test" defaultPools="blogdb-1,blogdb-2">
        <rule name="rule1">
            <parameters>ID</parameters>
            <expression><![CDATA[ abs(hash(id)) mod 2 = 0 ]]></expression>
            <defaultPools>blogdb-1</defaultPools>
        </rule>
        <rule name="rule2">
            <parameters>ID</parameters>
            <expression><![CDATA[ abs(hash(id)) mod 2 = 1 ]]></expression>
            <defaultPools>blogdb-2</defaultPools>
        </rule>
     </tableRule>
     <tableRule name="EVENT" schema="test" defaultPools="blogdb"/>
</amoeba:rule>
```

Xml tableRule.rule.parameters tag 介绍：

parameters 这儿指的是 sql 表达式里面的 where 条件中的 cloumn 字段。

比如：rule1 中参数 ID。 是 select * from message where **id**=12

多个 parameters 采用逗号分割

根据上面的数据切分规则：

1、当 select * from message where id=1 的时候，amoeba 将从 blogdb-2 中获取数据反馈给客户端。

2、当 select * from message where id in(1,2,3,4,5) 那么 amoeba 将从 blogdb-1、blogdb-2 同时发起请求，并且合并结果，然后将数据反馈给客户端。

Amoeba 提供很多数据切分规则函数，也支持非常复杂的数据切分规则。比如：

1、ID <= 1000000

2、ID between 1000001 and 2000000

3、ID>4 or CREATE_TIME between to_date('2008-11-12 00:00:00.0000') and to_date('2008-12-10 00:00:00.0000')

具体 amoeba 所支持的规则默认函数列表请参阅 sqljep

Amoeba 性能调优

1、内存参数设置

JVM option	Meaning
-Xms	initial java heap size
-Xmx	maximum java heap size
-Xmn	the size of the heap for the young generation
-Xss	the stack size for each thread

一般在 server 端的应用程序上面设置-Xms 跟 -Xmx 的值一致将有效提升应用程序的性能。

-Xms 小于等于-Xmx 的值。

比如修改 amoeba 启动脚本：DEFAULT_OPTS="-Xmx512m -Xms512m -Xmn100m -Xss1024k"

有些用户反映无法修改线程堆栈大小，则有些是受操作系统控制，linux 则可以通过

命令行执行 或者/etc/profile 文件中添加: ulimit -s 2048

2、如何提升 amoeba 并发响应速度。

- **多线程配置**

这个问题经常被提及到。Amoeba 响应速度主要取决机器性能跟 amoeba 的配置以及网络条件。

在 amoeba.xml 文件中 server 端的配置里面：

readThreadPoolSize –用于处理客户端连接发送过来的数据，跟数据库服务器返回的数据的线程数量这个可以根据客户端连接数量来调整。

clientSideThreadPoolSize –在读线程读完客户端的请求数据包以后，这个线程将会接手处理具体的业务逻辑（比如：parse sql、query route）

serverSideThreadPoolSize –在 server 端读取数据库返回的数据包,合并多数据库返回的数据，将数据包发送到客户端。

- **网络参数配置**

netBufferSize –参数主要设置 SendBufferSize、ReceiveBufferSize 大小。

1、SendBufferSize 属性设置在 SocketChannel.Write 方法的调用中期望发送的字节数。实际上，

此属性的操作对象是为发送操作所分配的网络缓冲区空间。网络缓冲区应至少与应用程序缓冲区同样大小，这样才能确保在一次操作中就能存储和发送所需的数据。使用 SendBufferSize 属性设置此大小。如果应用程序要发送批量数据，需为 Write 方法提供足够大的应用程序缓冲区。如果网络缓冲区小于提供给 Write 方法的数据量，则对 Write 方法的每次调用都将多次执行网络发送操作。通过确保网络缓冲区至少与应用程序缓冲区同样大小，可获得更大的数据吞吐量。

2、ReceiveBufferSize 属性获取或设置您希望在接收缓冲区中为每次读操作存储的字节数。实际上，此属性的操作对象是为接收传入数据所分配的网络缓冲区空间。网络缓冲区应至少与应用程序缓冲区同样大小，这样才能确保在调用 SocketChannel.Read 方法时所需的数据是可用的。使用 ReceiveBufferSize 属性设置此大小。如果应用程序将要接收批量数据，应为 Read 方法提供非常大的应用程序缓冲区。如果网络缓冲区小于在 Read 方法中所请求的数据量，您将无法在一次读取操作中检索到所需的数据量。这将导致增加对 Read 方法的调用次数，进而增加系统开销。

3、TcpNoDelay

如果禁用延迟，则为 true；否则为 false。默认值为 false。

如果 NoDelay 为 false，则直到 TcpClient 收集到相当数量的输出数据之后，它才会通过网络发送数据包。由于 TCP 段中系统开销的数量，发送少量数据时效率比较低。但是，也存在这样情况：您可能要发送极少量的数据或者期望立即从您发送的每个数据包得到响应。您的决定应取决于网络效率与应用程序要求之间的相对重要性。

Thanks

About author

author：struct chen 目前就职于阿里巴巴(中国)网络技术有限公司 从事分布式消息系统、分布式应用、多层框架设计、规则引擎开发框架研究，以及 java 2d mmorpg 框架。

可以通过 siru.chen@alibaba-inc.com 与我联系

Amoeba 下载地址: http://www.sf.net/projects/amoeba