E2dynamo 设计与实现

Erlang Easy Dynamo similar key-value storage system

litaocheng@gmail.com

http://code.google.com/p/e2dynamo

2008-12-20

Outline

- Dynamo
  - 介绍 (Introduction)
  - 关键算法 (Algorithms)
- E2dynamo
  - 介绍 (Introduction)
  - 设计 (Design)
  - 实现 (Implementation)
  - Roadmap

Dynamo：介绍

概述

- key-value 存储（数据大小 < 1M)
- 分布式，所有节点角色相同，负载均衡
- 接口简单， get/put (基于 HTTP)
- 快速响应（99.9% 请求 300ms 内完成)，大规模并发 (75K query/sec)
- 构建在一个可信任的网络
- 高容错，某些节点不可访问时，系统依旧可用
- 数据一致性，发生冲突时，客户端最后解决
- 系统易扩展，方便添加节点

Dynamo：介绍

应用场景

- amazon 电子平台应用
  - 用户购物车信息
  - 会话 (session) 管理
  - 商品的内容信息
- 三种常见应用 (根据冲突解决和读取需求划分)
  - 客户端根据逻辑解决冲突 (如购物车应用)
  - 时间最新原则选择数据 (如 session 的管理)
  - 高速数据读取 (可以作为 cache 应用)

Dynamo：介绍

其他相关联产品

- RDBMS 关注数据一致性， Dynamo 关注数据可用性
- Memcached 提供 cache 服务，不保证数据总是可用，数据具有临时性， Dynamo 提供持久性存储，快速响应，高可用性
- 与基于 DHT(chord ， Pastry) 的文件存储产品相比， Dynamo 的节点了解系统中大部分节点信息，只需 0 或 1 hop 即可到达目标节点，而不需迭代查询
- GFS (google file system) 包含一个中心化的 master 存储一些元信息，通过 chunk server 存储具体的数据。 Dynamo 没有中心化节点，所有节点完全相同平等
- CouchDB 提供半结构化数据存储，基于 HTTP 接口， Dynamo 数据为 key-value 数据形式，保证数据可用性

Dynamo: 关键算法

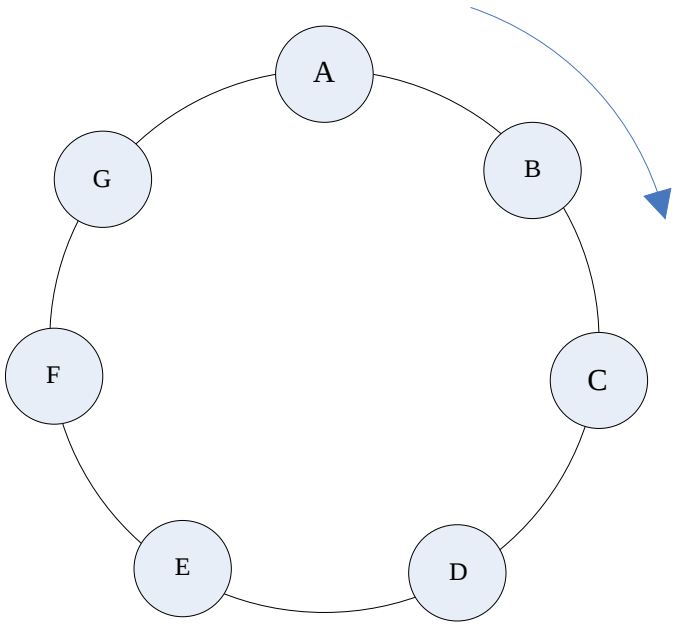
Outline

- Consistent Hashing
- 数据备份
- 数据一致性 (Quorum)
- 数据冲突处理 (Vector Clock)
- put/get API 执行
- Ring 空间的划分
- 数据同步 (Merkle Tree)

Dynamo: 关键算法

Consistent Hashing

- 每个节点拥有一个 ID，所有节点组成一个闭合的 Ring
- 根据数据的 Key 生成对应 ID (如 md5(key))，此 ID 映射到节点所组成的 Ring 上
- 数据存储在沿 Hashing Ring 顺时针方向第一个节点上
- 优点：新的节点插入时，只是影响临近的节点，系统振荡较小
- 如 (A, B] 之间数据存储在 B 上
- B 为 C 的前驱 (predecessor), D 为 C 的后继 (successor)



Dynamo: 关键算法

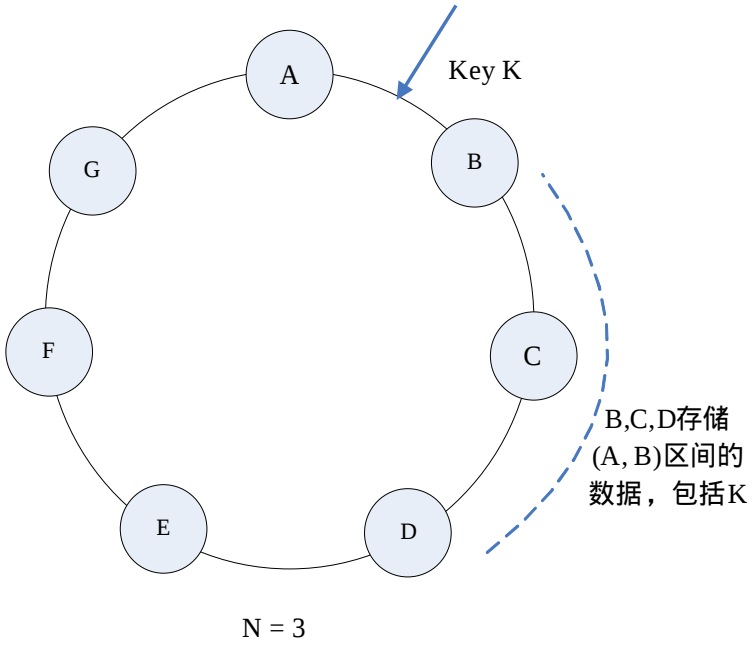
数据备份 (Replication)

- 为提高数据可用性， Dynamo 中每个 Key 对应数据在 N 个节点上进行备份 (通常 N=3)
- 在 Ring 上， Key 顺时针方向 N 个节点为其 Preference List
- 如果引入 Virtual Node 机制， List 中多个 Node 可能指向同一物理节点，为了使数据在系统中合理分布， List 中每个 Node 均指向不同的物理节点 (余下部分， Node 代表虚拟节点 , Physical Node 代表物理节点)
- 为了防止某个节点不可用， Preference List 中包含 Node 数大于 N
- Preference List 中选取一个 Node 作为 Coordinator(通常为 Preference List 中第一个节点)
- Coordinator 将数据备份在自身及 Preference List 中其它 N-1 个 Successor Node 上

Dynamo: 关键算法

数据备份 (Replication)

- 每个 Node 负责其前 N 个 Node 到自身的 Key 区间内的数据
- 如右图， D 负责 (A, B],(B, C], (C, D] 三个 Key 区间的数据



Dynamo: 关键算法

数据一致性 (Quorum)

- 定义
  - N ： 系统中数据的备份数
  - R： 读取操作最小成功数
  - W： 写操作最小成功数
  - 要求： R + W　　> N
- 配置
  - R+W ＞N，保证读操作至少能读取到一个最新数据
  - 读写速度受性能最差的节点影响
  - R 和 W 越小，系统的响应越快
  - W 越大，数据的一致性，可用性越强
  - 通常： N=3 ， R=W=2

Dynamo: 关键算法

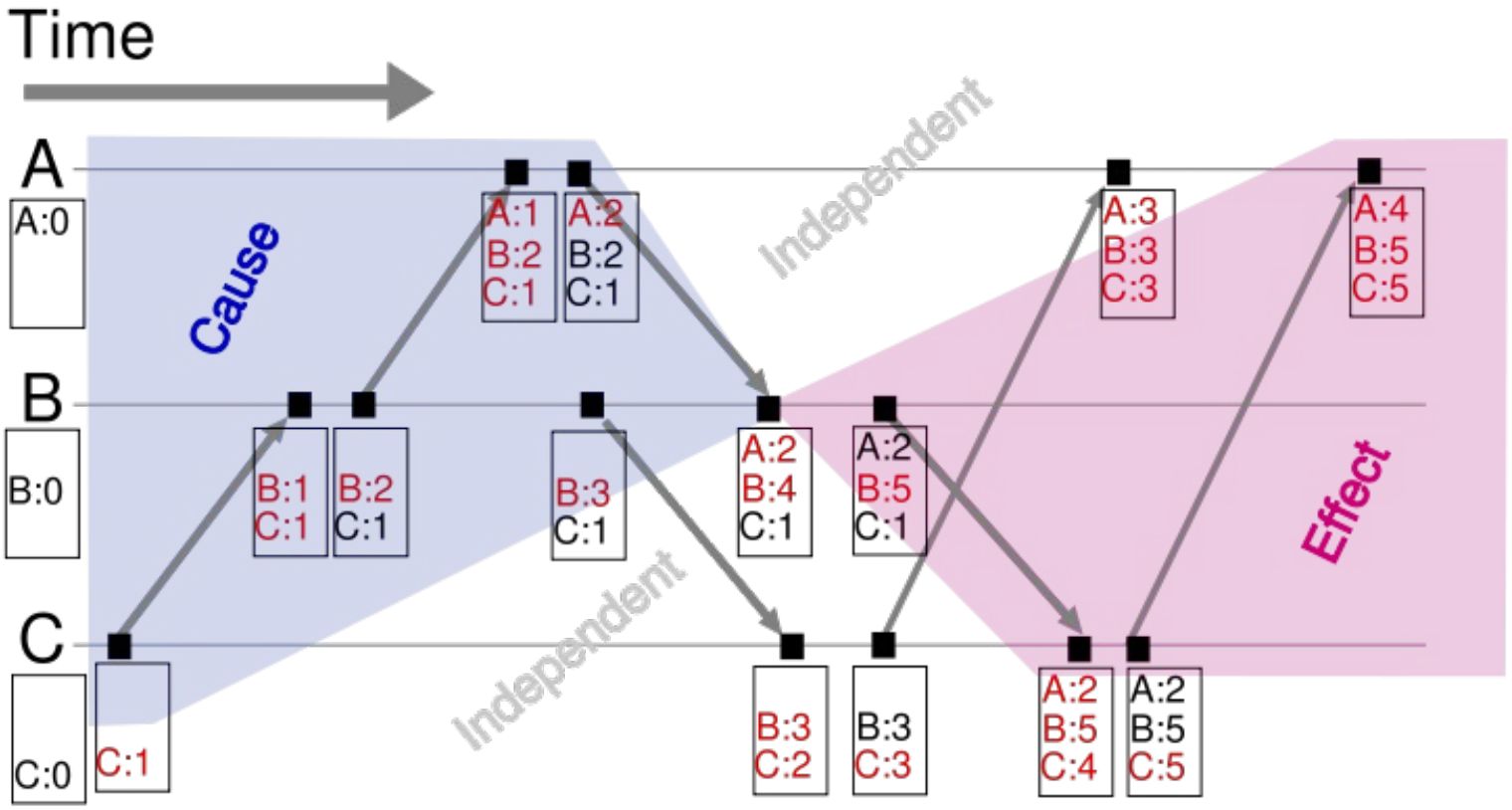
数据冲突处理 (Vector Clock)

- 在分布式环境中表示对象或事件的逻辑关系
- 包含多个 (name, counter) 数据 (用 Erlang 表示：[{atom(), non_neg_integer()}])
- 分布系统中每个消息都附加 vector clock 信息
- 每当某个节点内部处理一个事件时，其将自身对应 counter 加 1
- 节点发送一个消息前，将其自身对应的 counter 加 1
- 节点收到一个消息时，其将自身对应的 counter 加 1，同时根据消息中的 vector clock 信息更新所有的其他 counter

Dynamo: 关键算法

数据冲突处理（Vector Clock）

- 如果 clock A 中所有的 counter 计数小于等于 clock B, 则 A 是 B 的祖先



Dynamo: 关键算法

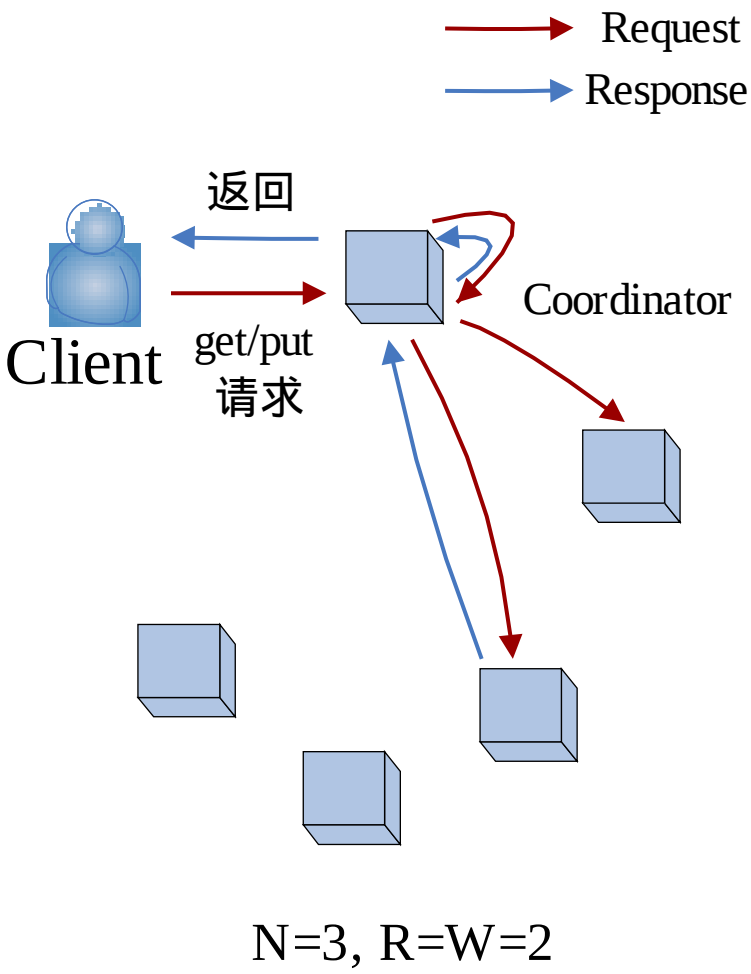
put/get API 执行

- Client

  向 Coordinator 发送请求

- Coordinator

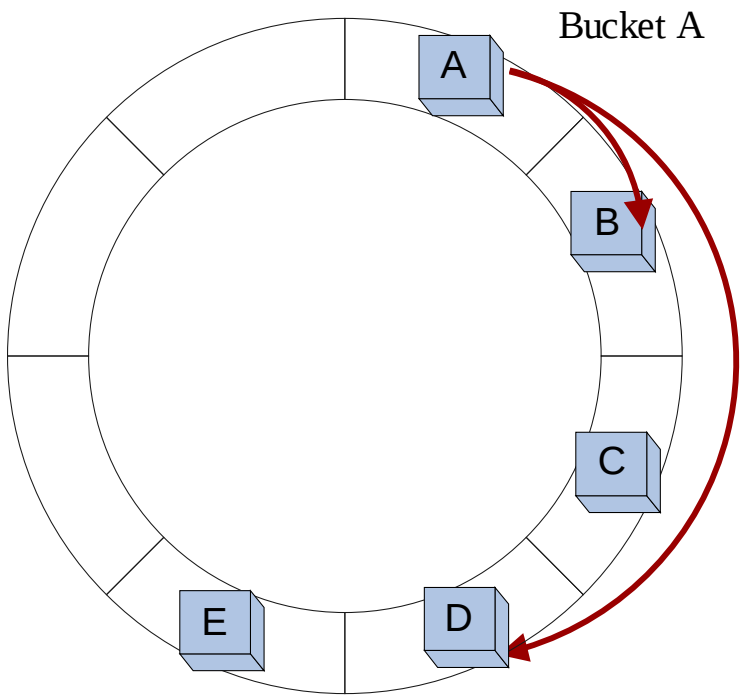
  1. 从 preference list 中选择 N 个可用节点
  2. 向 N 个节点发送请求
  3. 等待 R(get) 或 W(put) 个成功应答，或者超时
  4. 对结果进行处理 (get 操作，需要合并 version 信息)
  5. 将结果返回给 client，执行完成



Dynamo: 关键算法

Ring 空间的划分

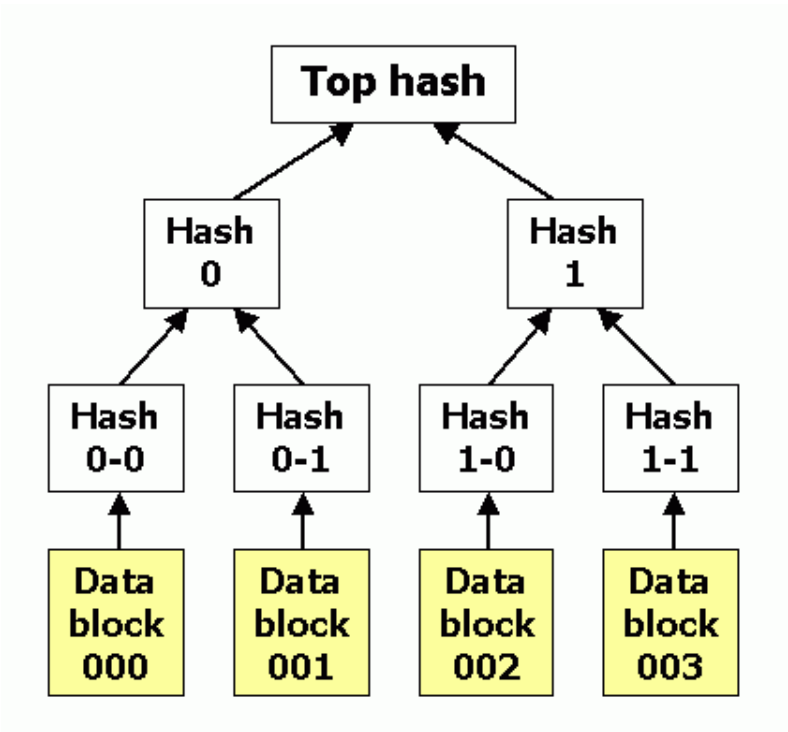
- 整个 Ring 被划分成 B 个大小相同的 Bucket
- 同一个 Bucket 上的数据落在相同的 N 个节点上
- 每个节点负责 N 个 Key 区间的数据
- 优点：便于节点间的数据同步 (参看后面 Merkle Tree 介绍)



N=3, Bucket A中的数据存储在B, C, D三个节点

Dynamo: 关键算法
数据同步（Merkle Tree)

- Merkle Tree 也称 Hash Tree， Leaf 节点是数据对应的 Hash 值， Parent 节点是其所有子节点的 Hash 值
- 如果两个 Merkle Tree Root 节点相同，则数据相同 , 否则从 Root 节点开始，由上到下判断节点是否相同
- 节点每个 Key 区间 (Bucket) 数据对应一个 Merkle Tree，可以快速的判断两个 Bucket 数据是否相同，如果数据不同，可以快速的定位差异，进行同步



E2dynamo: 介绍

- What is E2dynamo?
  - 基于 Erlang OTP 开发，实现一个简易的类似 amazon Dynamo 的系统
  - 提供基于 HTTP 的 API： put/get/del
  - 构建在 Erlang 自身的分布式机制之上，某些关键算法在 Dynamo 的基础上做了一些简化
  - 项目名称中 Easy 具有多重含义：简单；轻量；开发迅速
  - http://code.google.com/p/e2dynamo

E2dynamo: 介绍

- Why Erlang?
  - 基于消息，面向并发
  - FP 没有共享内存，不需要加锁解锁操作
  - 内建分布式支持，容错性强
  - OTP 丰富可信的编程模型
  - 久经考验的成功项目
  - 简洁，小巧，代码量少
  - 跨平台

E2dynamo: 设计

Outline

- 需求及原则
- 流程及关键算法
  - 节点加入 / 移除
  - get/put/del
  - 切分 Ring 空间
  - 错误检测机制
  - 数据冲突检测
  - 数据存储
  - Manager Server

E2dynamo: 设计

需求及原则

- 数据持久存储
- 数据高可用性，容错性强
- 快速的读写操作 (99.9% 请求再 500ms 内完成)
- 小规模集群 (推荐 1024 节点)
- 方便的加入删除节点
- 友好的管理界面
- 易安装部署

E2dynamo: 设计

节点加入 / 移除

- 节点从 Manager Server 获取 id， cookie， nodes 等信息
- 连接其它 Nodes
- 通过 Merkle Tree 同步 Bucket 数据
- 数据同步完成后，通知其 N 个后继节点移除数据，完成后节点加入成功
- 节点移除与此相反

E2dynamo: 设计

get/put/del(每个节点均可以处理请求)

- get(key)
  - 根据 Key 获取对应的 Coordinator
  - 向 Preference List 中前 N 个可用 Node 发送 get 请求
  - 等待 R 个成功返回，否则超时
  - 对结果进行 version 处理，返回给 Client
- put(key, context, value)
  - 根据 Key 从 Preference List 中挑选性能好的节点作为 Coordinator

E2dynamo: 设计

get/put/del(每个节点均可以处理 api)

- Coordinator 根据 context 来更新本节点的 vector clock 信息
- 向 preference list 中的前 W-1 个可用节点发送新的 value 和 vectotr 信息
- 等待 W-1 个成功返回，否则超时
- 结果返回给 Client

- del(key)
  - 同 put 相似，只是节点进行数据的删除

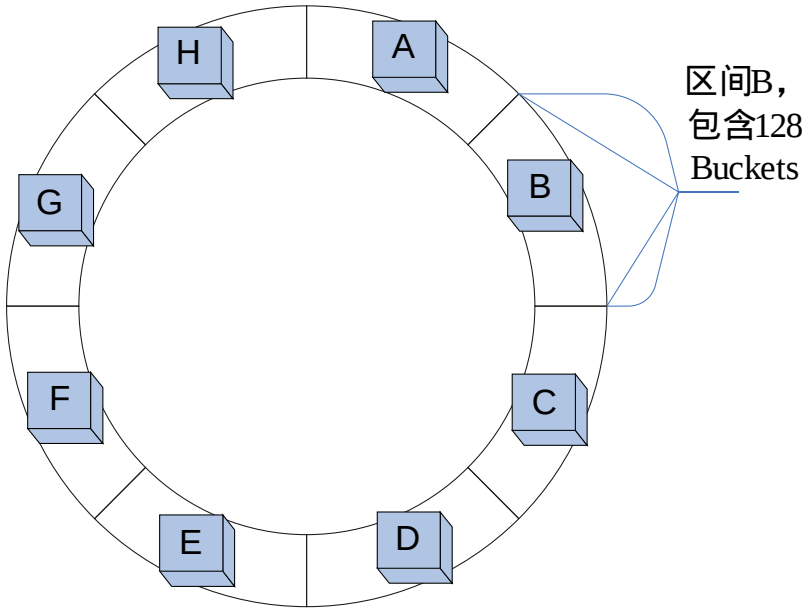
E2dynamo: 设计
切分 Ring 空间

- Ring 被切分成 Q 个大小相同的 Bucket
- 同一个 Bucket 的数据存储在相同的 Node 上
- 节点总数为 S, S < Q，部署整个系统时设定 Q，默认 Q=1024
- 节点 ID 通过 Manager Server 进行均匀分配，保证整个系统负载均衡
- 系统中初始节点数目 S=8，则每个节点负责 128 个 Bucket
- 节点管理的 Buckets 分为 :incharge Buckets 和 replica Buckets。节点与其第一个前驱节点之间的 Buckets 为 incharge Buckets，节点与其它前驱节点间的 Buckets 为 replica Buckets

E2dynamo: 设计

切分 Ring 空间

- 新节点加入时，根据 Node ID 切分原有节点管理的 Buckets，从后继节点同步其负责的 Buckets (incharge Buckets 和 replica Buckets)



Q=1024, S=8, 每个节点负责128个Buckets

区间 B 中的 Buckets 为节点 C 的 incharge Buckets，区间 H,A 中的 Buckets 为 C 的 replica Buckets

E2dynamo: 设计

错误检测机制

- 基于 Erlang 内建的分布式机制实现错误检测
- 通过 net_kernel:monitor_nodes/2 监控 nodes，当 node 加入退出时，调用者收到 {nodeup, Node, Info} 和 {nodedown, Node, Info} 消息
- 对于节点故障时间小于 T 为临时故障，大于 T 则转换为永久故障
- 没有采用 Dynamo 类似的 gossip 协议

E2dynamo: 设计
数据冲突检测

- 采用 Vector Clock 进行数据的冲突检测
- get
  - Coordinator 对 Node 返回数据进行 version 处理（依据 vector clock 或 time)
  - 如果数据一致，则将 Value 返回给 Client
  - 如果数据冲突，则将所有数据返回给 Client，由 Client 进行解决
- put
  - Coordinator 更新数据对应的 Vector Clock 信息，将数据和新的 Vector Clock 信息发送给其它节点
  - 如果 put 请求对应的版本信息和本地冲突则返回失败，否则成功
  - version 冲突的比例应该再一个较小的范围内

E2dynamo: 设计

数据存储

- 支持多种存储方式（可以自定义存储模块）
- Mnesia（默认存储方式）

  数据量大时，涉及 Fragmentation
- Mysql

E2dynamo: 实现

Apps

- Node

  E2dynamo 节点，处理具体请求，实现数据存储，备份，同步

- Manager Server

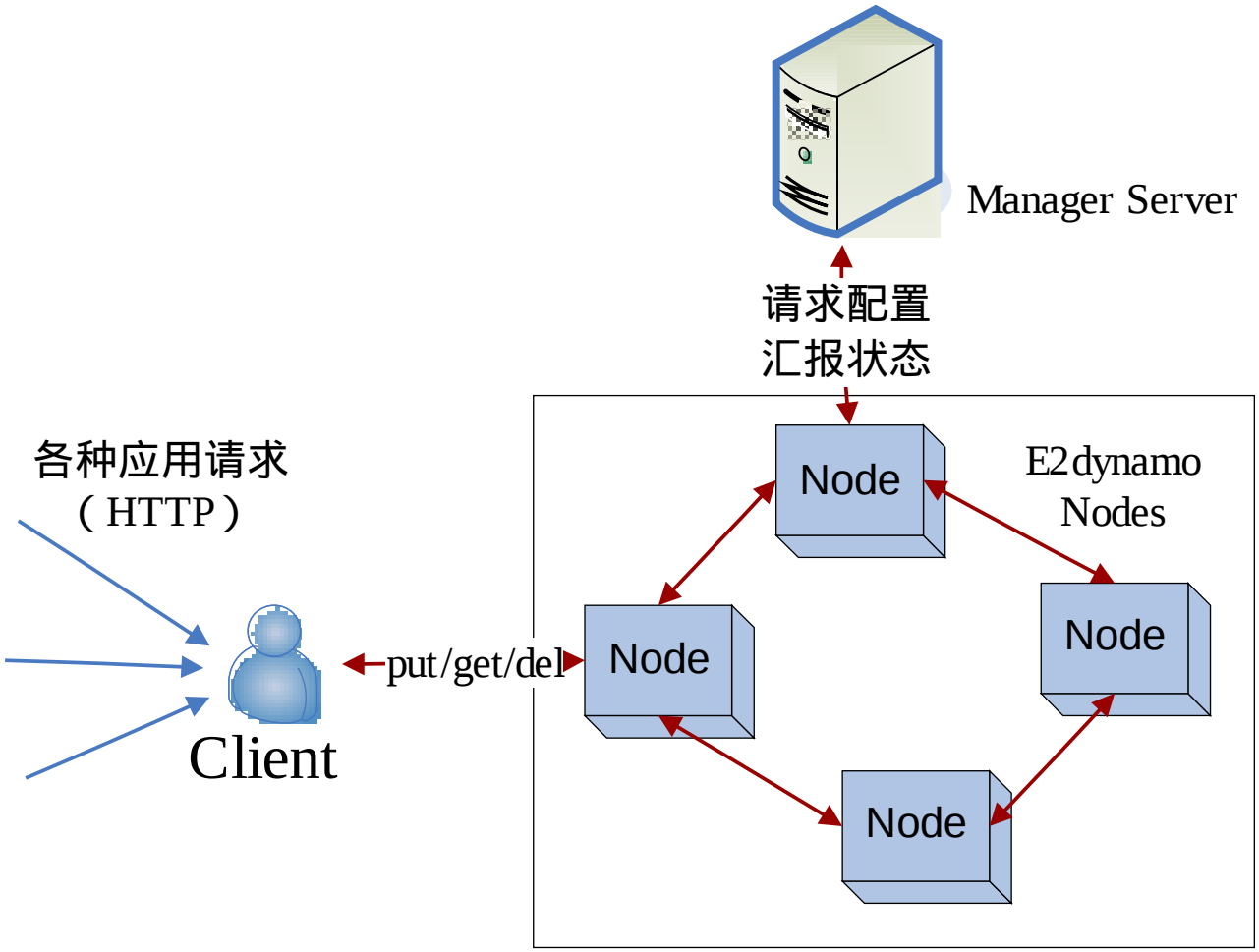
  管理所有的 Node ，添加，删除节点，配置下发，基于 HTTP 协议，采用 mochiweb 或其它 web 开发工具

- Client

  提供 HTTP 接口，接入 E2dynamo 系统，采用 mochiweb

E2dynamo: 实现

交互图



E2dynamo: 实现

Node

模块	Behaviour	含义
e2d.erl	application supervisor	app 和 supervisor callback module
e2d_comm.erl	-	负责节点之间通讯，基于 erlang 分布式机制
e2d_config.erl	gen_server	基于 http 从 cofigure server 获取配置
e2d_coordinator.erl	-	协调调度 client request
e2d_membership.erl	gen_server	管理维护系统中所有节点
e2d_node.erl	gen_server	本地节点信息
e2d_node_sup.erl	gen_event	监控节点事件 , 通知其它 event handler
e2d_server.erl	gen_server	处理各种 api 请求及其他节点的请求

E2dynamo: 实现

Node

模块	Behaviour	含义
e2d_store.erl	gen_server	处理各种数据操作 : 添加 / 删除 / 更新
e2d_store_mnesia.erl	gen_server	采用 Mnesia 进行数据的存储管理
e2d_sync.erl	-	利用 Merkle Tree 对数据进行同步
e2d_util.erl	-	相关的负责模块
vclock.erl	-	Vector Clock 模块
merkerl.erl	-	Merkle Tree 模块

E2dynamo: 实现

Manager Server

- 初期采用 mochiweb 实现一个简单的 http 管理界面，提供配置下发，节点状态监控，节点添加删除
- 根据节点的请求动态返回配置文件

```
[{n, 3}, {w, 3}, {r, 3},
 {cookie, 'e2dynamo-random-cookie2432343423cfdslluwer'}, %
    cookie
 {id, 0}, % node id
 {name, 'node_a@test.toquick.com'}, % node name
 {api_timeout, 1000}, % the api timeout
 {buckets_number, 1024}, % the bucket number
 {nodes, ['node_a@test.toquick.com', 'node_b@test.toquick.com',
    'node_c@test.toquick.com', 'node_d@test.toquick.com']} % all
    the nodes].
```

E2dynamo: 实现

Client

- 基于 HTTP 接口 :GET/PUT/DELETE
- Client library 定期 (1 second) 从 node 中获取系统中的节点信息
- 根据收到的 api 请求中的 key 确定 Coordinator 节点，随后想起发送请求
- 等待 node 返回结果，通过 http 将数据返回给外部应用
- 模块
  - e2d_c.erl
  - e2d_c_nodes.erl
  - e2d_c_api.erl
  - e2d_c_httpd.erl

E2dynamo:Roadmap

1. 原型 (2009.1.10)
   - 16 个 Node
   - 99.9% 请求 1 秒内返回
   - 简单的节点管理界面
1. 初级产品 (2009.4.1)
   - 128 个节点
   - 99.9% 请求 500ms 返回
   - 节点管理界面
   - 通讯安全机制

E2dynamo:Roadmap

3. 发布版本 (2009.6.1)
   - 1024 个节点 (虚拟节点)
   - 持久运行测试
   - 支持虚拟节点
   - 完善的节点管理

E2dynamo

E2dynamo OpenSource!

http://code.google.com/p/e2dynamo

参考：

Amazon's Dynamo

Kai – OpenSource key-value store

Distributerl

E2dynamo

E2dynamo

结束

谢谢各位！

期待 Erlang 在国内蓬勃发展！