Distribute Key-Value Store

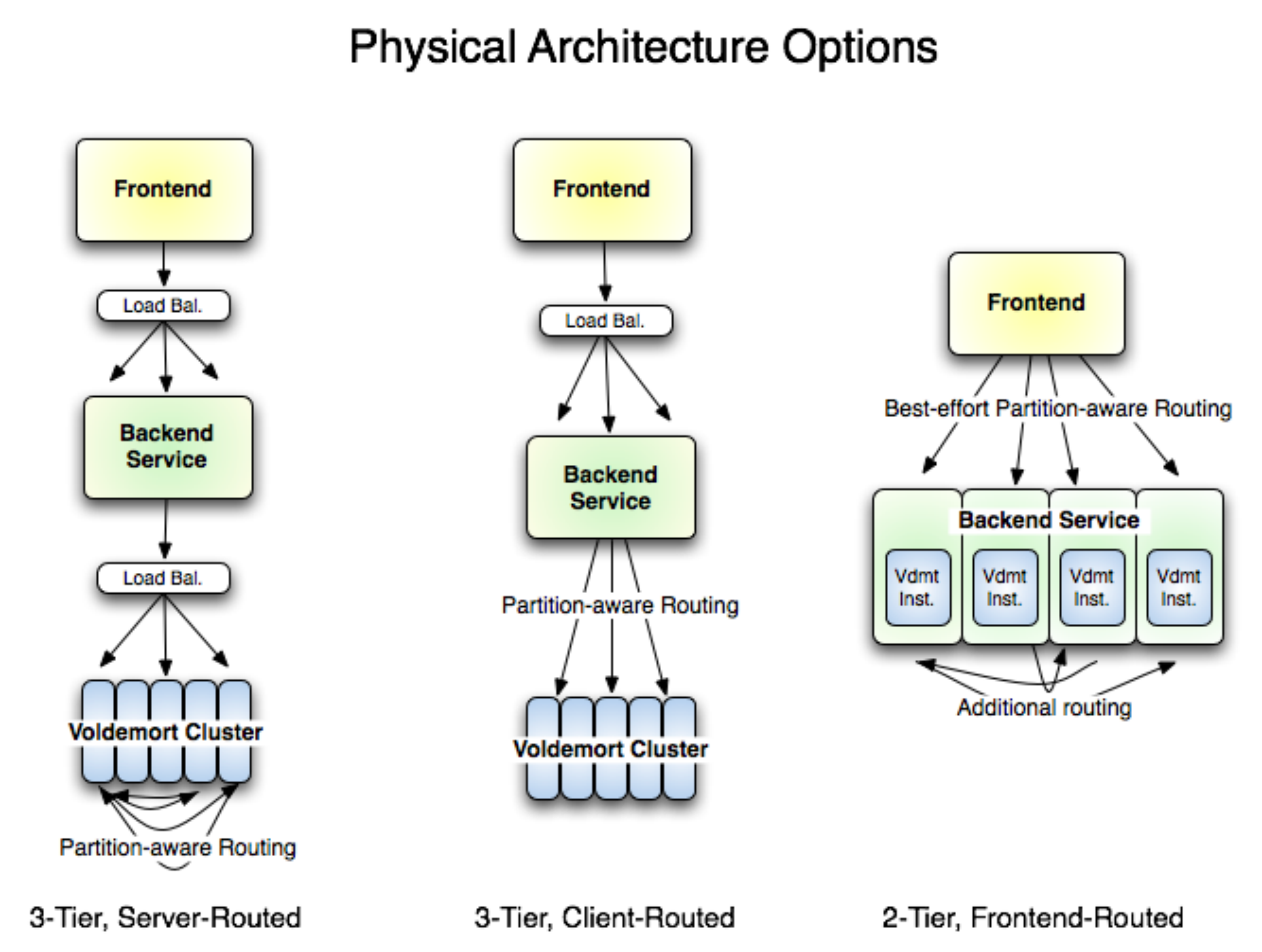
&Cassandra

Part I 基础介绍

Distribute KV Store 介绍

- KV Store
  - Key-Value Store 是一个存在了很久的数据库模式，
value = store.get(key) store.put(key, value) store.delete(key)
- KV Store的特点
  - 功能简单
  - 性能高
- Distribute KV Store
  - 把KV分布到多个Server上
  - 提供高并发访问和更大的容量
  - 最新的创新使DKV成为云计算的基础组件取得巨大成功

Distribute KV Store Architecture



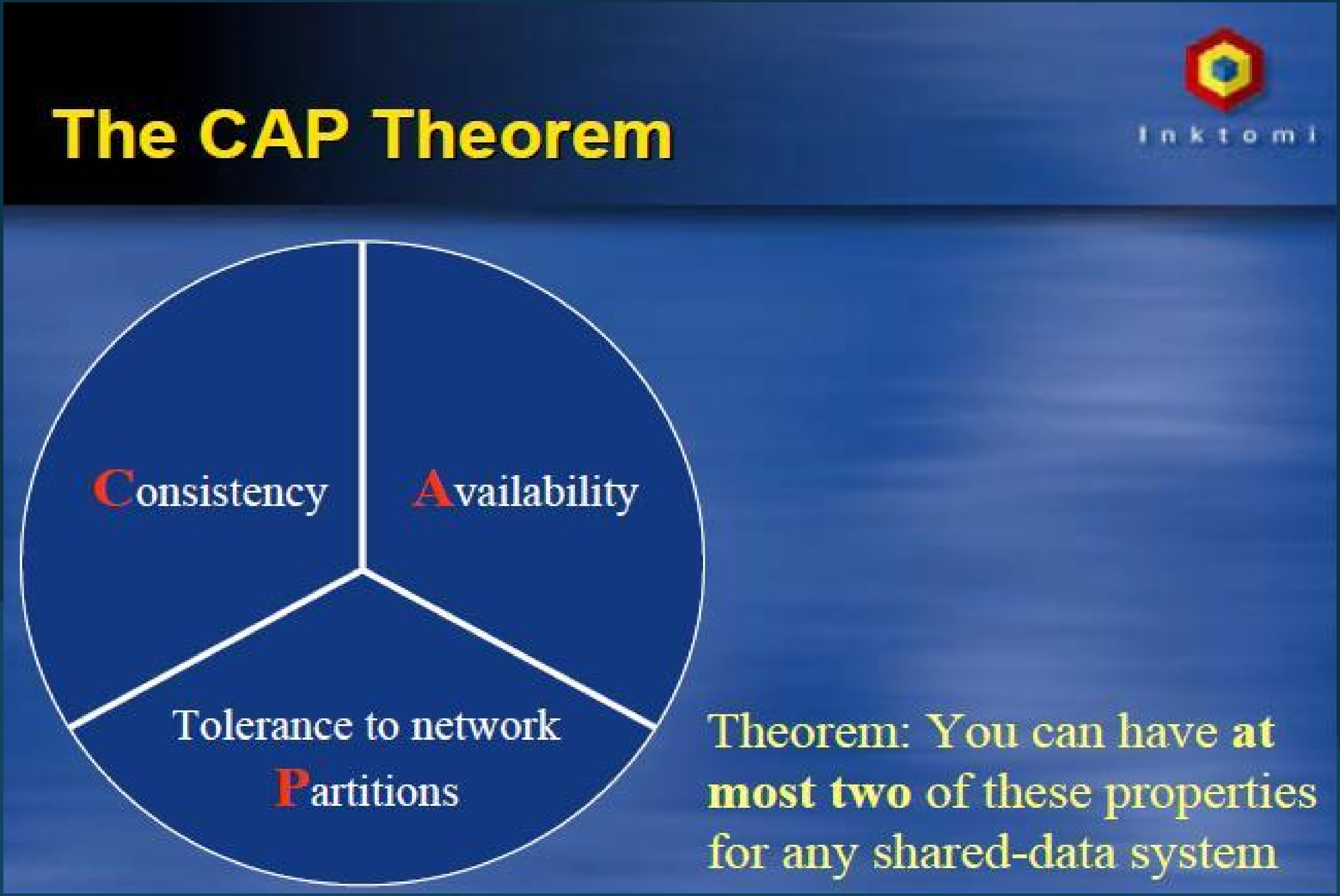
分布式的Key-Value Store和CAP原理

简单的Key-Value一般是嵌入到调用的应用程序中的，所以受限制于机器的内存和磁盘容量，不能服务于海量客户数据。尤其是现在的Web2.0时代，用户创建的数据非常多，不但用Key-Value Store难以存储，即便是使用RDBMS数据库系统，例如Oracle或者Mysql，也难以满足海量数据存放和海量并发读取的的要求，或者需要非常高的硬件配置和开二次发代价才能让系统正常工作。

在解决海量数据和海量并发双难题的过程中，业界的一些技术上领先的公司进展很大，象Amazon，Google，Yahoo都分别通过不同的方法解决了问题，他们的方法归结起来就是依靠一点：Distribute，Amazon出了著名的论文：<Dynamo: Amazon's Highly Available Key-Value Store>，并且在他们的云计算服务EC2/S3中实现了比Dyname更强大的Simple DB，Google组合了GFS和一系列其他的分布式算法，最终实现了BigTable，Yahoo既有分布式Key-Value Store的实现，也资助了类似于GFS/BigTable的Hadoop项目，到目前为止，Hadoop最大的生产集群依然由Yahoo建立并维护，多达1000 个Nodes.

CAP

在分布式系统中，有一个著名的CAP经验原理，说的是，对于一个分布式系统，consistency(数据一致性)，Availability(可用性)，Partitions(网络分隔)这三者只能同时满足其中的两个。



The CAP Theorem

inktomi

Consistency

Availability

Tolerance to network Partitions

Theorem: You can have **at most two** of these properties for any shared-data system

用CAP理论来解释RDBMS，它满足了Consistency和Availability，但正是因为如此，所以它在 Partition上就很难做得好。而对于很多的key-value形式的存储系统而言，它更强调的是Availability和Network Partition，所以在Consistency上做了弱化。例如，Amazon的Dynamo，它就不支持事务，只能提供Eeventra Consistency，为了 高可用性而牺牲了数据的一致性

“Dynamo targets applications that operate with weaker consistency
(the “C” in ACID) if this results in high availability”。

常见的DKV实现

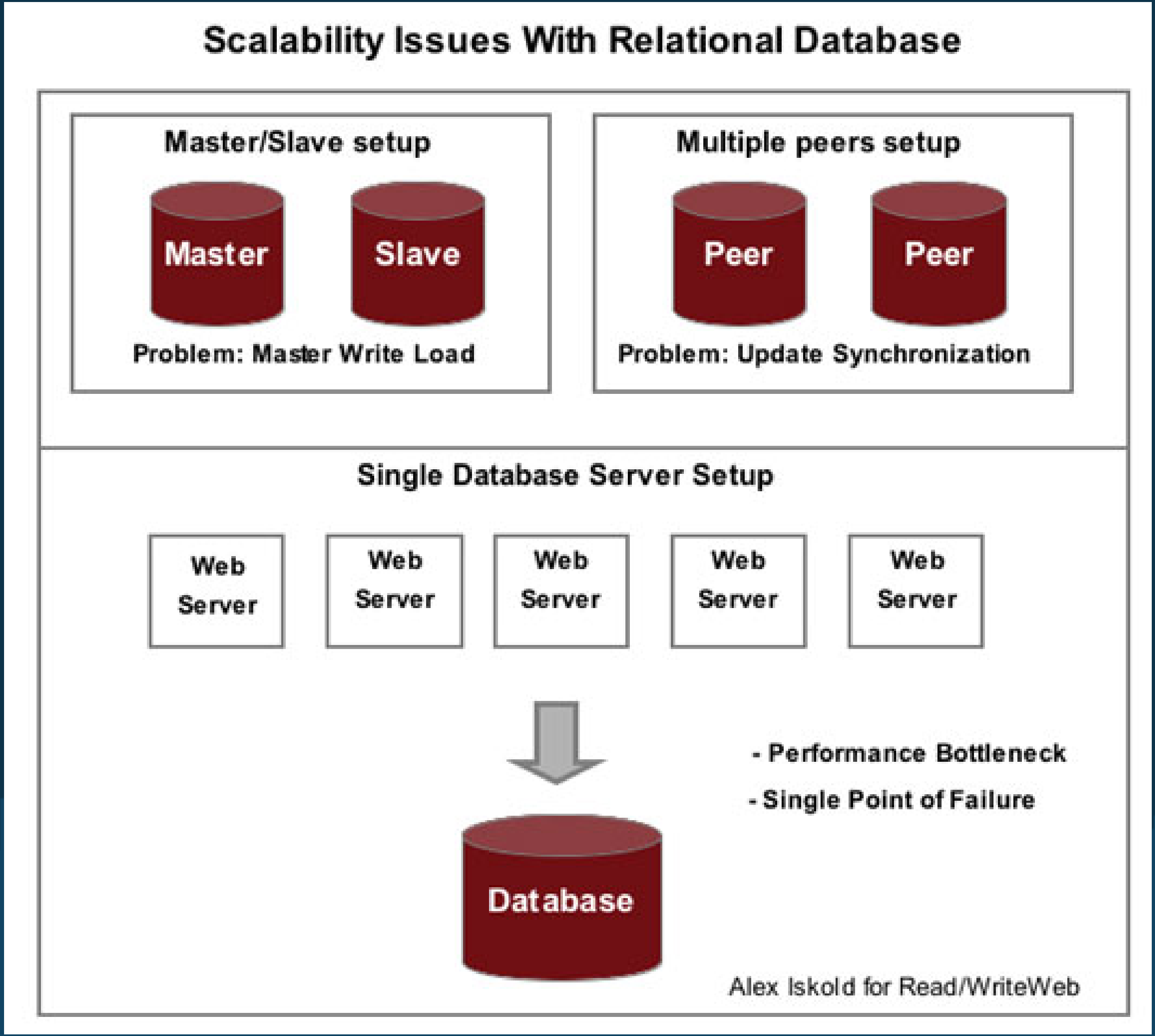
- Amazon Dynamo(下面有详细介绍）
  - Amazon Simple DB
  - Voldmont
  - Cassandra
- Google Big Table
  - HBase
  - Cassandra
  - Google App Engine
- Memcached
  - MemcacheDB

Amazon Dynamo

- Scalability Issues With Relational Databases

Tt is difficult to create redundancy and parallelism with relational databases

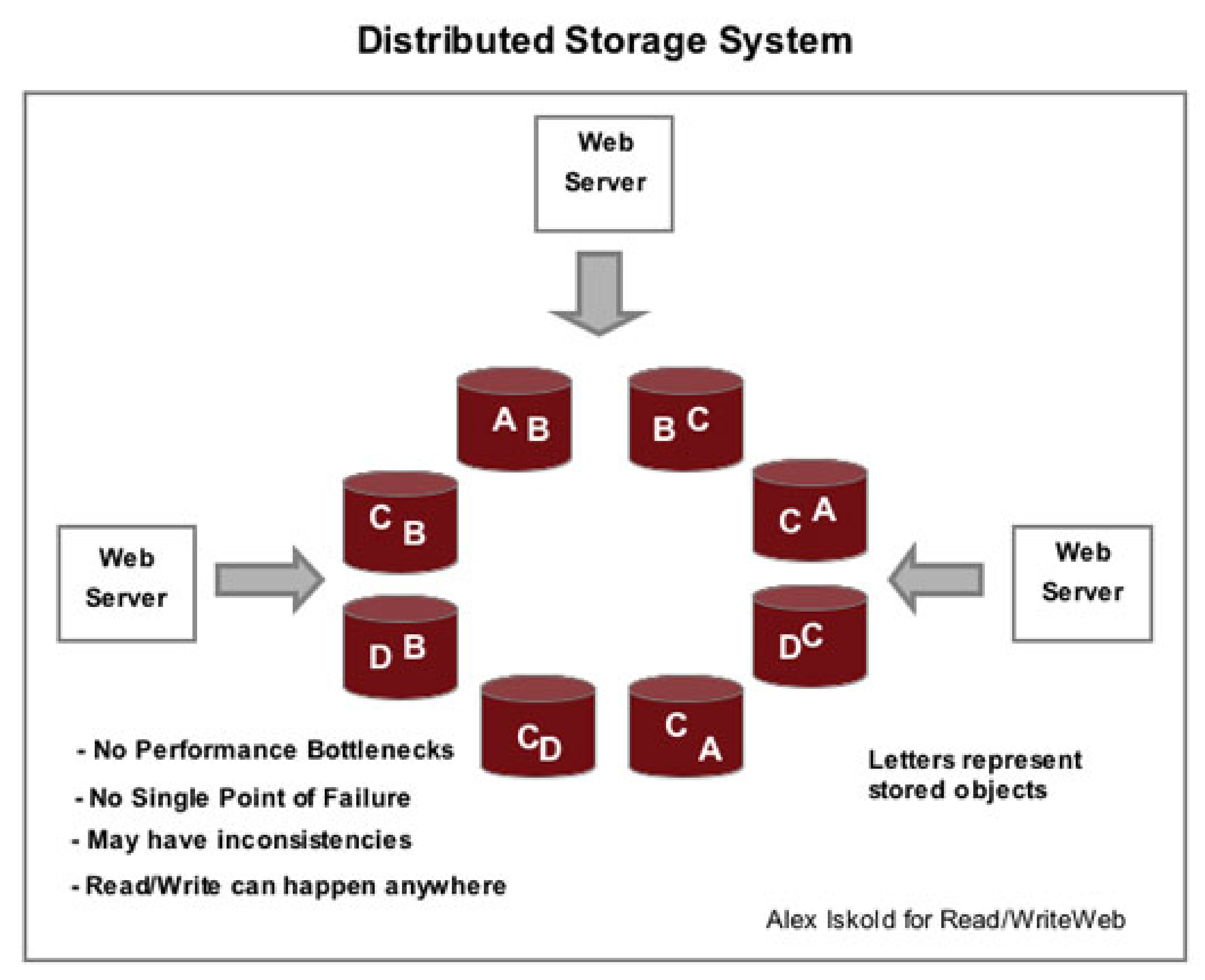
So as a relational database grows, it becomes a bottle neck and the point of failure for the entire system.


Scalability Issues With Relational Database
Master/Slave setup
Master
Slave
Problem: Master Write Load
Multiple peers setup
Peer
Peer
Problem: Update Synchronization
Single Database Server Setup
Web Server
Web Server
Web Server
Web Server
Web Server
Database
- Performance Bottleneck
- Single Point of Failure

Alex Iskold for Read/WriteWeb

Dynamo - A Distributed Storage System

- Dynamo is a distributed storage system
- the data is made redundant, so each object is stored in the system multiple times.
- Dynamo is called an eventually consistent storage system



Distributed Storage System

- No Performance Bottlenecks
- No Single Point of Failure
- May have inconsistencies
- Read/Write can happen anywhere

Letters represent stored objects

Alex Iskold for Read/WriteWeb

How Dynamo Works

- Physical nodes are thought of as identical and organized into a ring.
- Virtual nodes are created by the system and mapped onto physical nodes, so that hardware can be swapped for maintenance and failure.
- The partitioning algorithm is one of the most complicated pieces of the system, it specifies which nodes will store a given object.
- The partitioning mechanism automatically scales as nodes enter and leave the system.
- Every object is asynchronously replicated to N nodes.
- The updates to the system occur asynchronously and may result in multiple copies of the object in the system with slightly different states.
- The discrepancies in the system are reconciled after a period of time, ensuring eventual consistency.
- Any node in the system can be issued a put or get request for any key.

Part II
Distributed Key-Value Store的
算法基础

Data partitioning and replication

- Data Partitioning:
  - Data needs to be partitioned across a cluster of servers so that no single server needs to hold the complete data set.
- Replication:
  - Put the data into S partitions (one per server) and store copies of a given key K on R servers.

Consistency

- Tolerate the possibility of inconsistency, and resolve inconsistencies at read time.
  - Two-Phase Commit — This is a locking protocol that involves two rounds of co-ordination between machines. It perfectly consistent, but not failure tolerant, and very slow.
  - Paxos-style consensus — This is a protocol for coming to agreement on a value that is more failure tolerant.
  - Read-repair — The first two approaches prevent permanent inconsistency. This approach involves writing all inconsistent versions, and then at read-time detecting the conflict, and resolving the problems. This involves little co-ordination and is completely failure tolerant, but may require additional application logic to resolve conflicts.
- Dynamo use versioning and read-repair.

Versioning

- A vector clock keeps a counter for each writing server, and allows us to calculate when two versions are in conflict, and when one version succeeds or preceeds another.

A vector clock is a list of server:version pairs:

[1:45,2:3,5:55]

The version indicates that the server was the "master" for that number of writes.

A version v1 succeeds a version v2 if for all i, v1i > v2i. If neither v1 > v2 nor v1 < v2, then v1 and v2 co-occur, and are in conflict.

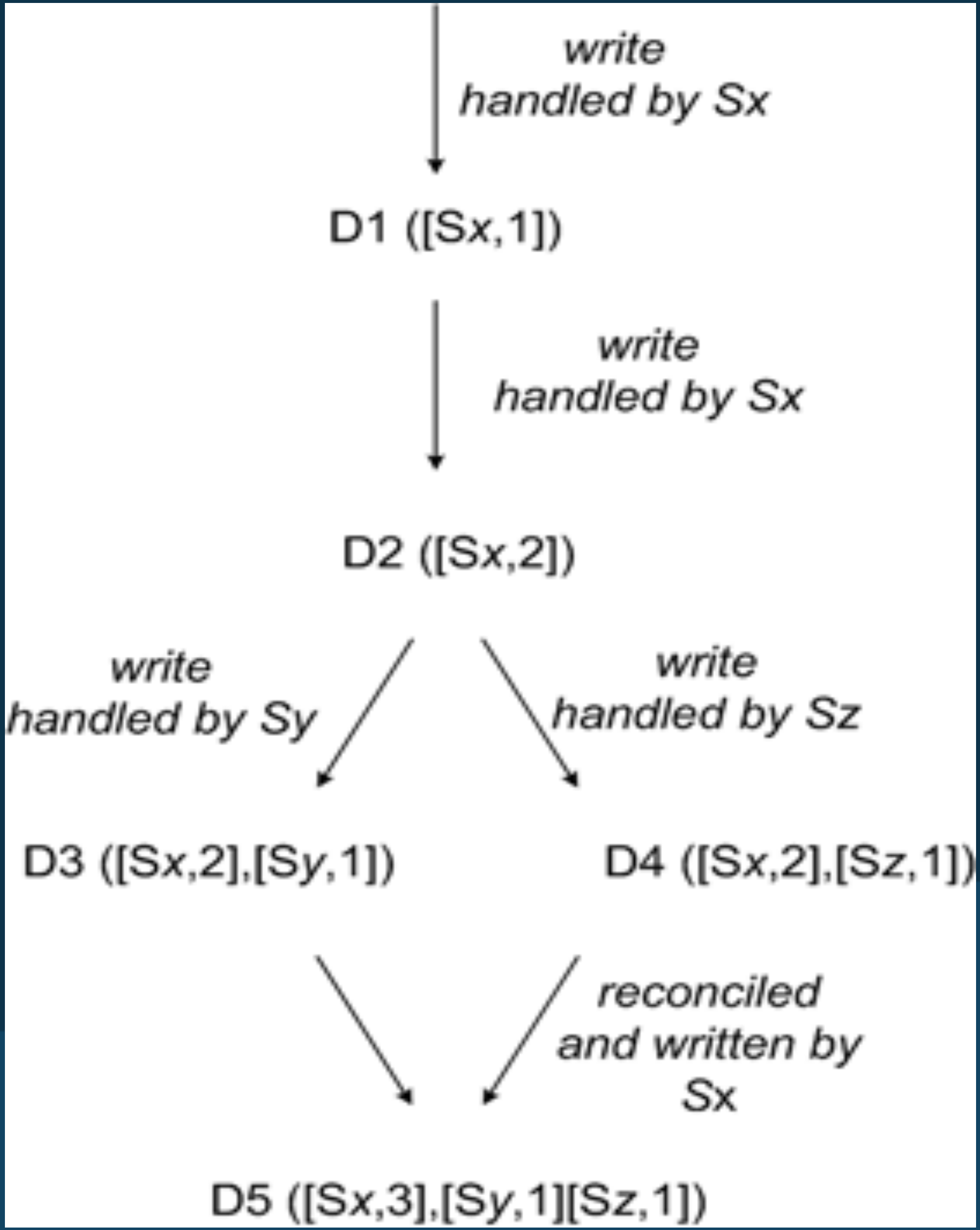
- Here is a simple example of two conflicting versions:

[1:2,2:1]

[1:1,2:2]

- vector clock versioning scheme defines a partial order over values where simple optimistic locking schemes define a total order.

Vector Clock Version



write
handled by Sx

D1 ([Sx,1])

write
handled by Sx

D2 ([Sx,2])

write
handled by Sy

write
handled by Sz

D3 ([Sx,2],[Sy,1])

D4 ([Sx,2],[Sz,1])

reconciled
and written by
Sx

D5 ([Sx,3],[Sy,1][Sz,1])

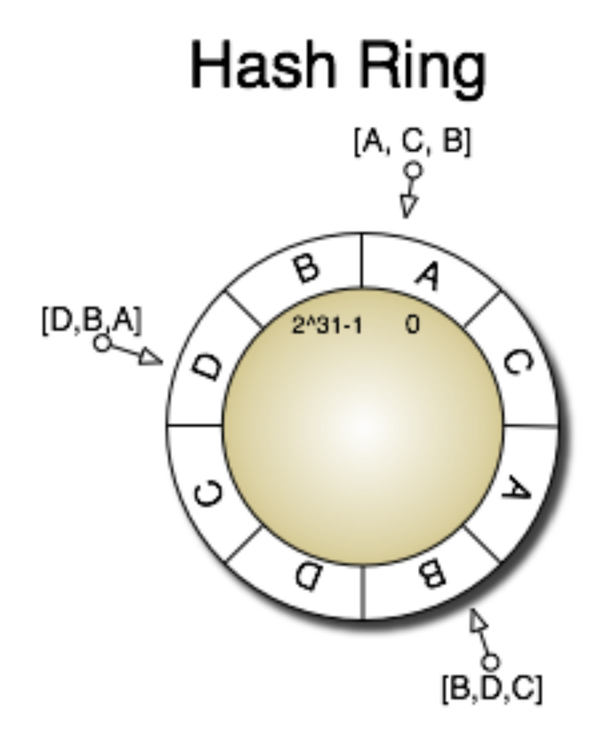
Data partitioning and replication

- Hash Partition
  - 使用某种Hash函数分布数据, 性能好, 简单， 缺点在于如果HashBacket发生改变， 会在大范围上导致rehash
- Consistent hashing
  - Consistent hashing is a technique that avoids these problems
  - When a new server is added to a cluster of S servers, only 1/(S+1) values must be moved to the new machine.

Consistense Hashing

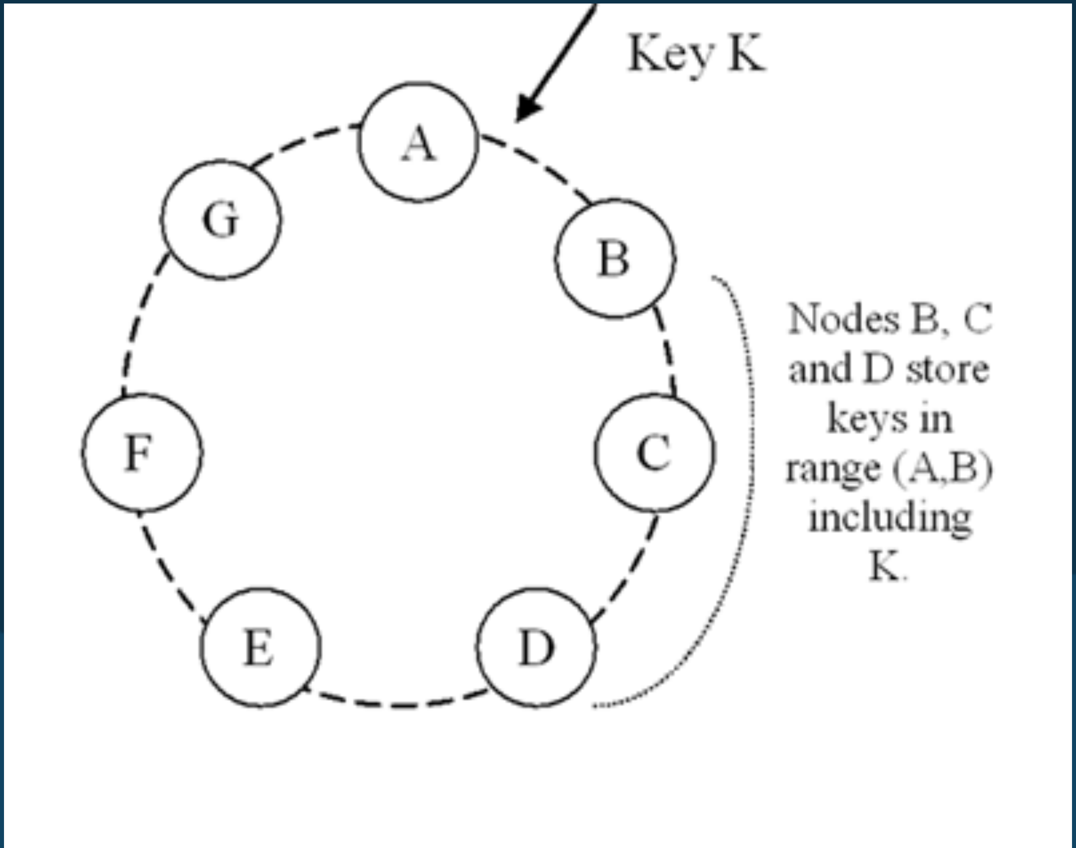
- we can see the possible integer hash values as a ring beginning with 0 and circling around to 2^31-1.
- This ring is divided into Q equally-sized partitions with Q >> S, and each of the S servers is assigned Q/S of these.
- A key is mapped onto the ring using an arbitrary hash function, and then we compute a list of R servers responsible for this key by taking the first R unique nodes when moving over the partitions in a clockwise direction.

- *The diagram below pictures a hash ring for servers A,B,C,D. The arrows indicate keys mapped onto the hash ring and the resulting list of servers that will store the value for that key if R=3.*



Replication

To achieve high availability and durability, Dynamo replicates its data on multiple hosts.



Others

- Handling Failures: Hinted Handoff
- Handling permanent failures: Replica synchronization
- HashTree(Merkely Tree)
- ...

http://s3.amazonaws.com/AllThingsDistributed/sosp/amazon-dynamo-sosp2007.pdf

Part III DKV的应用

Distributed Key-Value Store的典型应用

- Session Store

SessionID / Session Object

- User注册

UserID / Current Step / Preview Step

- 产品推荐

Dataware Hourse / Greenplum
针对每个用户提供推荐

Cassandra

http://incubator.apache.org/cassandra

Cassandra's Features:

- Cassandra is designed to be always available. Writes never fail. Two read paths are available: high-performance "weak" reads and quorum reads. See ThriftInterface.
- Cassandra has a rich data model allowing efficient use for many applications beyond simple key/value.
- Data is automatically replicated to multiple nodes for fault-tolerance. There is support for implementing strategies that replicate across multiple data centers.
- Elasticity: new nodes can be added to a running cluster while minimizing disruption to existing data.
- Consistency: Cassandra follows the "eventually consistent" model but includes sophisticated features such as Hinted Handoff and Read Repair to minimize inconsistency windows.
- Reads and writes in Cassandra are guaranteed to be atomic within a single ColumnFamily.
- Support for versioning and conflict resolution (with inbuilt policies like "last update wins").

Install

- Java1.6
- Apache Thrift
- package 0.4.1
- Only one configure file, very easy to install.

Client Code

```
... TTransport tr = new TSocket("localhost", 9160); TProtocol proto = new TBinaryProtocol(tr);
Cassandra.Client client = new Cassandra.Client(proto); tr.open(); String key_user_id = "1"; // insert
data long timestamp = System.currentTimeMillis(); client.insert("Keyspace1", key_user_id, new
ColumnPath("Standard1", null, "name".getBytes("UTF-8")), "Chris Goffinet".getBytes("UTF-8"), // read
single column ColumnPath path = new ColumnPath("Standard1", null, "name".getBytes("UTF-8"));
System.out.println(client.get("Keyspace1", key_user_id, path, ConsistencyLevel.ONE)); // read entire
row SlicePredicate predicate = new SlicePredicate(null, new SliceRange(new byte[0], new byte[0],
false, 10)); ColumnParent parent = new ColumnParent("Standard1", null); List<ColumnOrSuperColumn>
results = client.get_slice("Keyspace1", key_user_id, parent, predicate, ConsistencyLevel.ONE); for
(ColumnOrSuperColumn result : results) { Column column = result.column; System.out.println(new
String(column.name, "UTF-8") + " -> " + new String(column.value, "UTF-8")); } tr.close(); ...
```

```
service Cassandra {
  # retrieval methods
  ColumnOrSuperColumn get(string keyspace, string key,ColumnPath column_path,
ConsistencyLevel consistency_level=1)

  list<ColumnOrSuperColumn> get_slice(string keyspace, string key, ColumnParent
column_parent, SlicePredicate predicate, ConsistencyLevel consistency_level=1)

  map<string,ColumnOrSuperColumn> multiget(string keyspace, list<string> keys,  Co
column_path, ConsistencyLevel consistency_level=1)

  map<string,list<ColumnOrSuperColumn>> multiget_slice(string keyspace,
list<string> keys,  ColumnParent column_parent,  SlicePredicate predicate,
ConsistencyLevel consistency_level=1)

  i32 get_count(string keyspace, string key, ColumnParent column_parent,
ConsistencyLevel consistency_level=1)

  # range query: returns matching keys
  list<string> get_key_range( string keyspace, string column_family, string
start="",  string finish="", i32 count=100,  ConsistencyLevel consistency_level=1)
              throws (1: InvalidRequestException ire, 2: UnavailableException ue),

  # modification methods
  void insert(string keyspace, string key,  ColumnPath column_path, binary value,
i64 timestamp, ConsistencyLevel consistency_level=0)

  void batch_insert(string keyspace,  string key, map<string,
```

Cassandra Source Code

- 遵循SEDA模型，Server可以处理大量的并发
- Memory Table
- Serialize all data change action
- 大量使用异步Callback模型

性能测试结果

Thanks

FAQ