Motivation

Even the slightest outage has significant financial consequences and impacts customer trust.

The platform is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world.

Persistent state is managed in the face of these failures drives the reliability and scalability of the software systems.

Motivation (Cont’d)

Build a distributed storage system:
- Scale
- Simple: key-value
- **Highly available** (sacrifice consistency)
- Guarantee Service Level Agreements (SLA)

System Assumptions and Requirements

- **Query Model**
- **ACID Properties**
- **Efficiency**
- **Other Assumptions**

Query Model

- simple read and write operations to a data item that is uniquely identified by a key.
- Most of Amazon’s services can work with this simple query model and do not need any relational schema.
- targets applications that need to store objects that are relatively small (usually less than 1 MB)

ACID Properties

- **Atomicity** (原子性):
  - 每个事务被视为全部或者什么也没有——不是提交 (commit) 就是中止 (abort)
- **Consistency** (一致性):
  - 确保事务不允许系统到达一个不准确的逻辑状态。即使在发生系统故障时，约束和规则也必须得到承兑。
- **Isolation** (隔离性):
  - 并发事务与其他未完成 (incomplete) 事务的更新操作分隔
- **Durability** (持久性):
  - 确保该事务的作用持续存在 (即使发生系统故障)。
ACID Properties (Cont’d)

- Experience at Amazon has shown that data stores that provide ACID guarantees tend to have poor availability.
- Dynamo targets applications that operate with weaker consistency (the “C” in ACID) if this results in high availability.
- Dynamo does not provide any isolation guarantees and permits only single key updates.

Efficiency

- Latency requirements which are in general measured at the 99.9th percentile of the distribution.
- Average performance is not enough.

Other Assumptions

- Operation environment is assumed to be non-hostile and there are no security related requirements such as authentication and authorization.

Service Level Agreements (SLA)

- Application can deliver its functionality in bounded time.
  - Every dependency in the platform needs to deliver its functionality with even tighter bounds.
- Example:
  - Service guaranteeing that it will provide a response within 300ms for 99.9% of its requests for a peak client load of 500 requests per second.

Service-oriented architecture

- Sacrifice strong consistency for availability.
- Conflict resolution is executed during read instead of write, i.e. “always writeable.”
Design Consideration (Cont’d)

- Incremental scalability.
- Symmetry
  - Every node in Dynamo should have the same set of responsibilities as its peers
  - In our experience, symmetry simplifies the process of system provisioning and maintenance.
- Decentralization.
  - In the past, centralized control has resulted in outages and the goal is to avoid it as much as possible.
- Heterogeneity.
  - This is essential in adding new nodes with higher capacity without having to upgrade all hosts at once.

Design Consideration (Cont’d)

- “always writeable” data store where no updates are rejected due to failures or concurrent writes.
- an infrastructure within a single administrative domain where all nodes are assumed to be trusted.

Design Consideration (Cont’d)

- do not require support for hierarchical namespaces (a norm in many file systems) or complex relational schema (supported by traditional databases).
- built for latency sensitive applications that require at least 99.9% of read and write operations to be performed within a few hundred milliseconds.

System architecture

- Partitioning
- High Availability for writes
- Handling temporary failures
- Recovering from permanent failures
- Membership and failure detection

Partition

- Motivation
  - One of the key design requirements for Dynamo is that it must scale incrementally.
  - This requires a mechanism to dynamically partition the data over the set of nodes (i.e., storage hosts).

Partition (Cont’d)

- If there are many data, use cache to lower server load.
- Use hash function, build data to cache mapping
- If simple hashing function, when the range of hash is changed, much data must be stored in the new position
- Consistent hashing has 2 properties
  - When a new machine is added or removed, be moved to new position
  - Expected fraction of objects smallest
  - Different cache number small

Partition (Cont’d)
Consistent hashing: the output range of a hash function is treated as a fixed circular space or “ring”.

“Virtual Nodes”: Each node can be responsible for more than one virtual node.

Advantages of using virtual nodes:
- If a node becomes unavailable, the load handled by this node is evenly dispersed across the remaining available nodes.
- When a node becomes available again, or a new node is added to the system, the newly available node accepts a roughly equivalent amount of load from each of the other available nodes.
- The number of virtual nodes that a node is responsible can be decided based on its capacity, accounting for heterogeneity in the physical infrastructure.

Each data item is replicated at N hosts.

“Preference list”: The list of nodes that is responsible for storing a particular key.

A put() call may return to its caller before the update has been applied at all the replicas.
A get() call may return many versions of the same object.

Challenge: an object having distinct version sub-histories, which the system will need to reconcile in the future.
Solution: uses vector clocks in order to capture causality between different versions of the same object.

A vector clock is a list of (node, counter) pairs.
Every version of every object is associated with one vector clock.
If the counters on the first object’s clock are less-than-or-equal to all of the nodes in the second clock, then the first is an ancestor of the second and can be forgotten.

A vector clock is a list of (node, counter) pairs.
Every version of every object is associated with one vector clock.
If the counters on the first object’s clock are less-than-or-equal to all of the nodes in the second clock, then the first is an ancestor of the second and can be forgotten.
Vector clock

- In case of network partitions or multiple server failures, write requests may be handled by nodes that are not in the top $N$ nodes in the preference list causing the size of vector clock to grow.
- Dynamo stores a timestamp that indicates the last time the node updated the data item.
- When the number of (node, counter) pairs in the vector clock reaches a threshold (say 10), the oldest pair is removed from the clock.
- Further issue has not been thoroughly investigated.

Execution of get () and put () operations

- Two strategies to select a node:
  1. Route its request through a generic load balancer that will select a node based on load information.
  2. Use a partition-aware client library that routes requests directly to the appropriate coordinator nodes.

Execution of get () and put () operations (Cont’d)

- The advantage of the first approach is that the client does not have to link any code specific to Dynamo in its application
- The second strategy can achieve lower latency because it skips a potential forwarding step.

配额设置

- 三个关键参数 $(N,R,W)$
- $N$ 指数据对象将被复制到 $N$ 台主机上
- $N$ 在 Dynamo 实例级别配置，协调器将负责把数据复制到 $N-1$ 个节点上。
- $N$ 的典型值设置为 3.

配额设置

- 复制中的一致性，采用类似于 Quorum 系统的一致性协议实现。这个协议有两个关键值：$R$ 与 $W$。
- $R$ 代表一次成功的读取操作中最小参与节点数量
- $W$ 代表一次成功的写操作中最小参与节点数量
- $R + W > N$，则会产生类似 quorum 的效果。
- 该模型中的读(写)延迟由最慢的 $R(W)$ 复制决定，为得到比较小的延迟，$R$ 和 $W$ 有的时候的和又设置比 $N$ 小。

配额设置

- $(N,R,W)$ 的值典型设置为 $(3, 2, 2)$，兼顾性能与可用性。$R$ 和 $W$ 直接影响性能、扩展性、一致性
- 如果 $W$ 设置为 1，则一个实例中只要有一个节点可用，也不会影响写操作
- 如果 $R$ 设置为 1，则只要有一个节点可用，也不会影响读请求
- $R$ 和 $W$ 值过小则影响一致性，过大也不好，这两个值要平衡。对于这套系统的典型的 SLA 要求 99.9% 的读写操作在 300ms 内完成。
**Hinted handoff**

- Assume N = 3. When A is temporarily down or unreachable during a write, send replica to D.
- D is hinted that the replica is belong to A and it will deliver to A when A is recovered.
- Again: “always writeable”

**Replica synchronization**

- **time-stamped anti-entropy protocol**
  - Replica tells others what the request is, and get the request status.
  - Any two replicas exchange the request to make the time stamp maximum.

**Replica synchronization (Cont’d)**

- **summary vector Vi**
  - Vij = t: i has received j’s request before t.
- **Log vector**
  - A log entry for each update request
  - Use the request’s time stamp.

**Replica synchronization (Cont’d)**

- Avoid log infinity, each Replica only stores the largest time after the request.
- Use **acknowledgement vector**
- Acknowledgement vector
  - Vij = t: i knows j has received the request before t.
Replica synchronization (Cont’d)

A and B confirm: A received 9 moment before all operations.
A and B confirm: B received 9 moment before all operations.
A and B confirm: C received 4 moment before all operations.

Structure of Merkle tree:
- A hash tree where leaves are hashes of the values of individual keys.
- Parent nodes higher in the tree are hashes of their respective children.

Replica synchronization (Cont’d)

Advantage of Merkle tree:
- Each branch of the tree can be checked independently without requiring nodes to download the entire tree.
- Help in reducing the amount of data that needs to be transferred while checking for inconsistencies among replicas.

Summary of techniques used in Dynamo and their advantages

<table>
<thead>
<tr>
<th>Problem</th>
<th>Technique</th>
<th>Advantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>Vector clocks with reconciliation during reads</td>
<td>Version size is decoupled from update rate.</td>
</tr>
<tr>
<td>High Availability for writes</td>
<td>Gossip protocol and hinted transferral</td>
<td>Provides high availability and durability guarantees when some of the replicas are not available.</td>
</tr>
<tr>
<td>Handling temporary failures</td>
<td>Anti-entropy using Merkle trees</td>
<td>Maintains divergent replicas in the background.</td>
</tr>
<tr>
<td>Recovering from permanent failures</td>
<td>Gossip-based membership protocol and failure detection</td>
<td></td>
</tr>
<tr>
<td>Membership and failure detection</td>
<td>Anti-entropy using Merkle trees</td>
<td></td>
</tr>
</tbody>
</table>

Implementation

- Java
- Local persistence component allows for different storage engines to be plugged in:
  - Berkeley Database (BDB) Transactional Data Store: object of tens of kilobytes
  - MySQL: object of > tens of kilobytes
  - BDB Java Edition, etc.

Performance

- Guarantee Service Level Agreements (SLA)
  - Latencies exhibit a clear diurnal pattern (incoming request rate)
  - Write operations always result in disk access.
  - Affected by several factors such as variability in request load, object sizes, and locality patterns
Improvement

- A few customer-facing services required higher levels of performance.
- Each storage node maintains an object buffer in its main memory.
- Each write operation is stored in the buffer and gets periodically written to storage by a writer thread.
- Read operations first check if the requested key is present in the buffer.

Improvement (Cont’d)

- lowering the 99.9th percentile latency by a factor of 5 during peak traffic.
- write buffering smooths out higher percentile latencies.

Improvement (Cont’d)

- a server crash can result in missing writes that were queued up in the buffer.
- To reduce the durability risk, the write operation is refined to have the coordinator choose one out of the N replicas to perform a “durable write”.
- Since the coordinator waits only for W responses, the performance of the write operation is not affected by the performance of the durable write operation.

Balance

- out-of-balance
  - If the node’s request load deviates from the average load by a value more than a certain threshold (here 15%)
  - Imbalance ratio decreases with increasing load
  - under high loads, a large number of popular keys are accessed and the load is evenly distributed.

Partitioning and placement of key

- the space needed to maintain the membership at each node increases linearly with the number of nodes in the system.
- the schemes for data partitioning and data placement are intertwined. It is not possible to add nodes without affecting data partitioning.

Partitioning and placement of key (cont’d)

- divides the hash space into Q equally sized partitions.
- The primary advantages of this strategy are:
  1. decoupling of partitioning and partition placement.
  2. enabling the possibility of changing the placement scheme at runtime.
Partitioning and placement of key (cont’d)

- divides the hash space into \( Q \) equally sized partitions
- each node is assigned \( Q/S \) tokens where \( S \) is the number of nodes in the system.
- When a node leaves the system, its tokens are randomly distributed to the remaining nodes.
- when a node joins the system it "steals" tokens from nodes in the system.

Strategy 3 achieves better efficiency
- Faster bootstrapping/recovery:
  - Since partition ranges are fixed, they can be stored in separate files, meaning a partition can be loaded by simply transferring the file avoiding random access needed to load specific items.
- Ease of archival:
  - Periodic archiving of the dataset is a mandatory requirement for most of Amazon storage services.
  - Archiving the entire dataset stored by Dynamo is simpler in strategy 3 because the partition files can be archived separately.

Coordination

- Dynamo has a request coordination component that uses a state machine to handle incoming requests. Client requests are uniformly assigned to nodes in the ring by a load balancer.
- An alternative approach to request coordination is to move the state machine to the client nodes. In this scheme client applications use a library to perform request coordination locally.

The latency improvement is because the client-driven approach eliminates the overhead of the load balancer and the extra network hop that may be incurred when a request is assigned to a random node.

Conclusion

- Dynamo is a highly available and scalable data store, used for storing state of a number of core services of Amazon.com’s e-commerce platform.
- Dynamo has been successful in handling server failures, data center failures and network partitions.

Dynamo is incrementally scalable and allows service owners to scale up and down based on their current request load.
- Dynamo allows service owners to customize their storage system to meet their desired performance, durability and consistency SLAs by allowing them to tune the parameters \( N, R, \) and \( W \).