Couchbase NoSQL Database

Pradeep Tummala, Database Engineer, PayPal Core Data Platform

AGENDA

- RDBMS v/s NoSQL
- Why Couchbase ?
- Cocuhbase concepts (stuff that matters)
- Data Modeling guidelines
- Applications of Couchbase
- Ops & Dev Tips
- Summary and Q & A

Speaker Qualifications

Currently Database Engineer @ PayPal

Has been working with Oracle Databases and UNIX for 12 years

Working on various NoSQL technologies for the past 3 years

Has worked on many Sharded applications – Both Oracle and NoSQL

http://www.linkedin.com/in/Pradeep-tummala

Decisions - (RDBMS vs NoSQL): a 4 x 4 matrix

Challenges of Traditional RDBMS

1. ACID overheads inhibit scalability
2. Lack of native sharding
3. Need for “schema before write”
4. Higher base cost for setup and scale up

Advantages of Traditional RDBMS

1. ACID is essential in many cases!
2. Complex data model support
3. Mature technology and ecosystem
4. Wider skill availability

Advantages of NoSQL

1. Highly (and quickly) scalable at relatively lower cost
2. Low latency, *scalable* K-V read/write
3. Flexible data model *1
4. Open source and Enterprise model

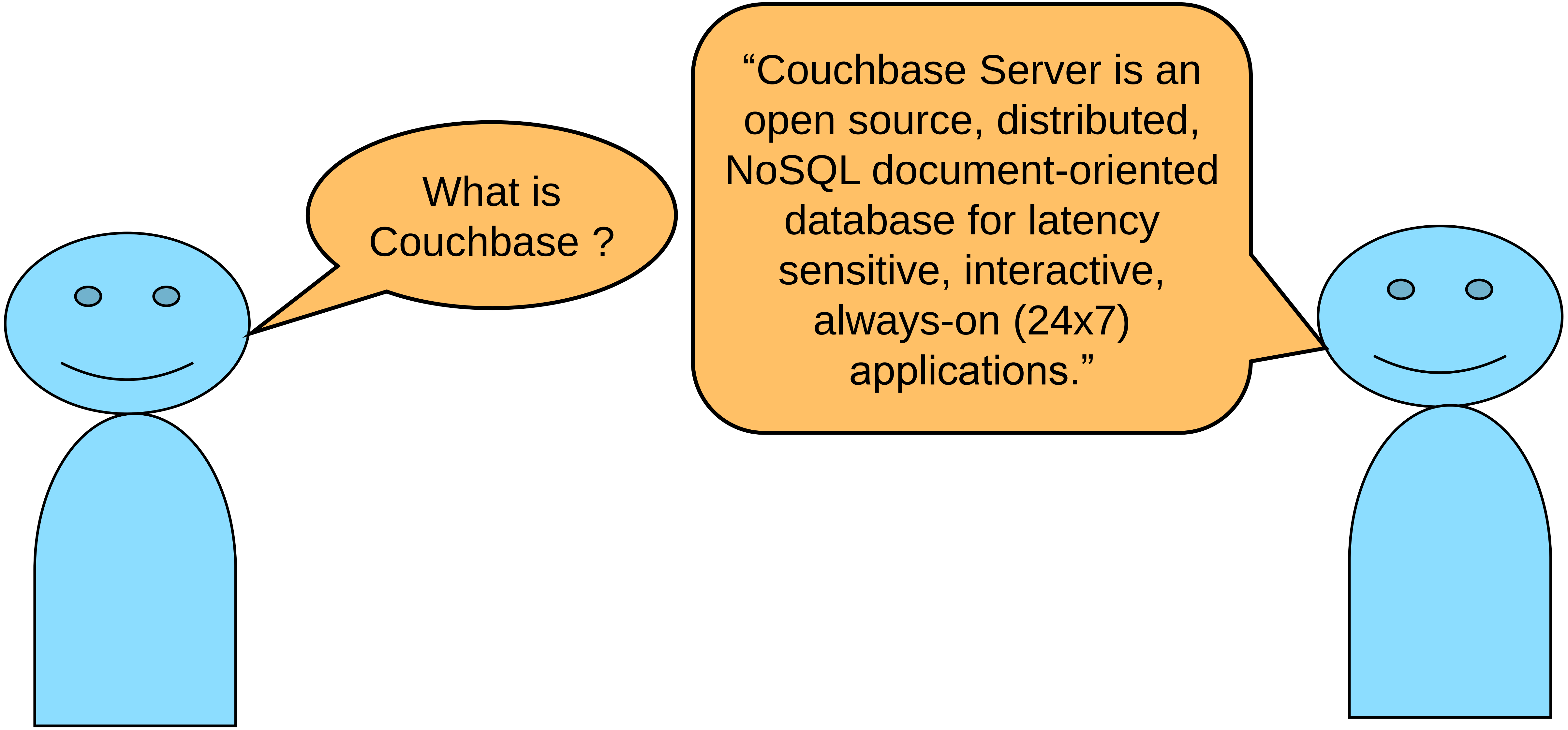
Challenges of NoSQL

1. Limited ACID and Transactions
2. CAP Theorem is real! *2
3. Technology evolving, maturing but fragmented landscape
4. Skillsets not widely available (yet!)

*1 – Establishing an initial data model is easy, evolving it is harder
*2 – This caveat applies to all distributed databases, whether RDBMS or NoSQL. Out of box though, NoSQL databases are distributed

Couchbase Footprint at PayPal

- PayPal uses a Polyglot of Database technologies
  - Oracle RDBMS
  - NoSQL – Couchbase, Cassandra, Aerospike, MongoDB, …
- Couchbase usage @ PayPal
  - One of the earliest NoSQL's adopted (2013)
  - Used primarily for low latency caching, cookie store, temporary token store, anamoly detection, config management, etc.
  - Mostly Couchbase 4.1.x, planned to move to 5.x+ this year (*POC in progress*)
  - Eight Couchbase cluster families, 250+ servers
- Database team organized by Architecture, Engineering, Delivery and Operations across multiple geographies, supporting polyglot technologies


What is Couchbase ?
“Couchbase Server is an open source, distributed, NoSQL document-oriented database for latency sensitive, interactive, always-on (24x7) applications.”

Why Couchbase?

Easy Scalability

Grow cluster without application changes, without downtime with a single click

Consistent High Performance

Consistent sub-millisecond read and write response times with consistent high throughput

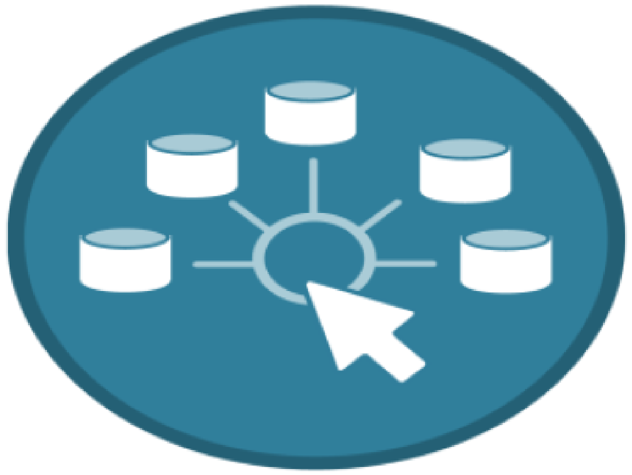
Always On 24x365

No downtime for software upgrades, hardware maintenance, etc.

Flexible Data Model

JSON Anywhere document model with no fixed schema.

Easy Scalability



Auto Sharding

No Manual Sharding

Database manages data movement to scale out - Not the user

XDCR

Database handles propagation of updates to scale across clusters and geos

Provides disaster recover / data locality

Single package Multi-service

Hugely simplifies management of clusters

Easy to scale clusters by adding any # of nodes

Consistent High Performance

Massive Concurrent Connections

Support a large number of users needed for interactive apps

Fine Grained Locking

Allows high concurrency and in turn high throughput via highly granular latches

Built-in Cache

No need of separate cache layer

Database manages actively used data

Hash Partitioning

Uniform data distribution

Uniform load distribution – NO hotspots

Always on 24x7 Capability

Online DB upgrades and maintenance	Online administrative operations	HA via Replication DR via XDCR
Online DB upgrades and HW maintenance Optimized swap operation to replace nodes	All admin operations online • Compaction • Indexing • Rebalance • Backup & Restore	• High availability using in-memory replication • Auto or manual failover • XDCR for disaster recovery

Flexible Data Model

Schema-less for structured / un/semi-structured data

Data with mixed structure better managed via JSON in a document DB than an RDBMS

Maintains Native object representation

Represent data as objects instead of shredding into rows and columns

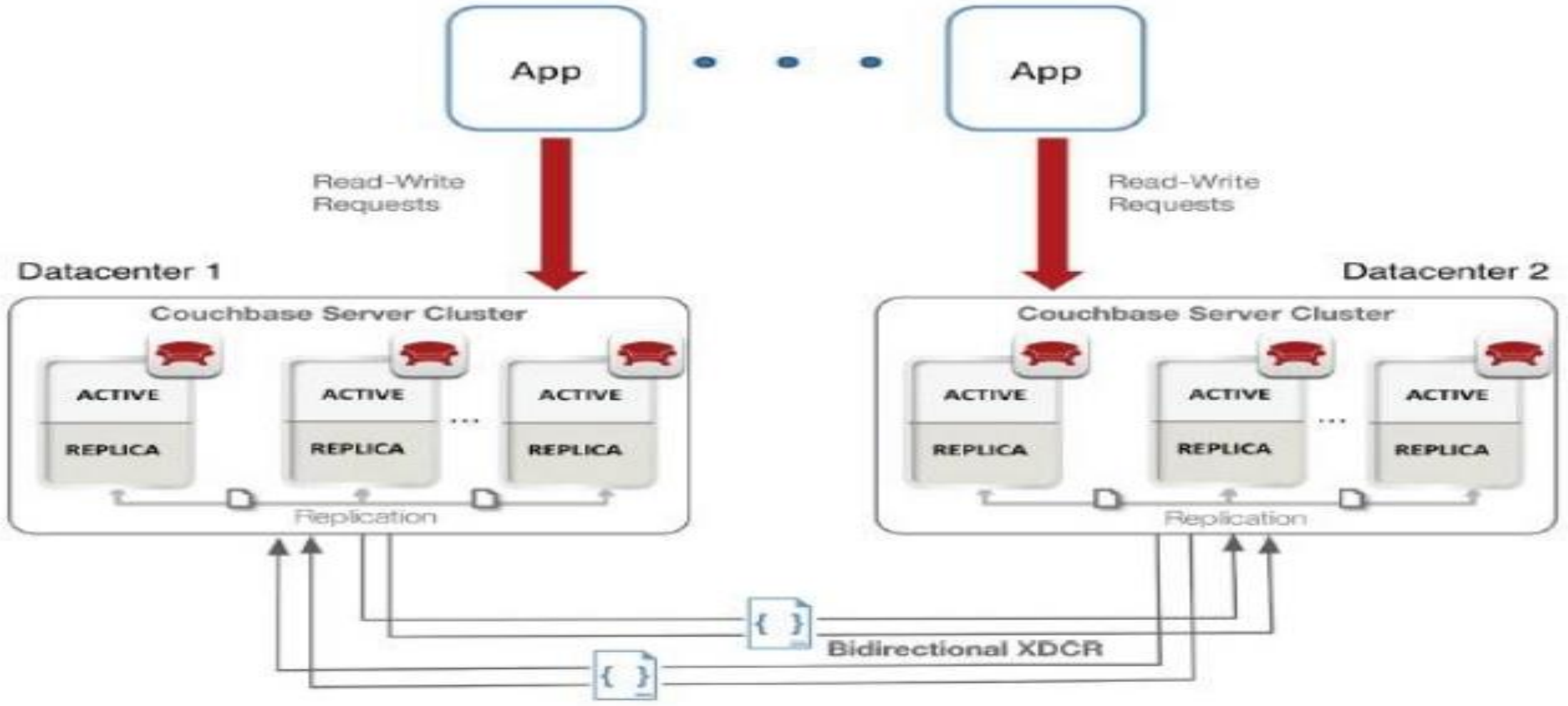
Create indexes on any attribute of a JSON document

Handles constantly changing data

Each document can have a different structure

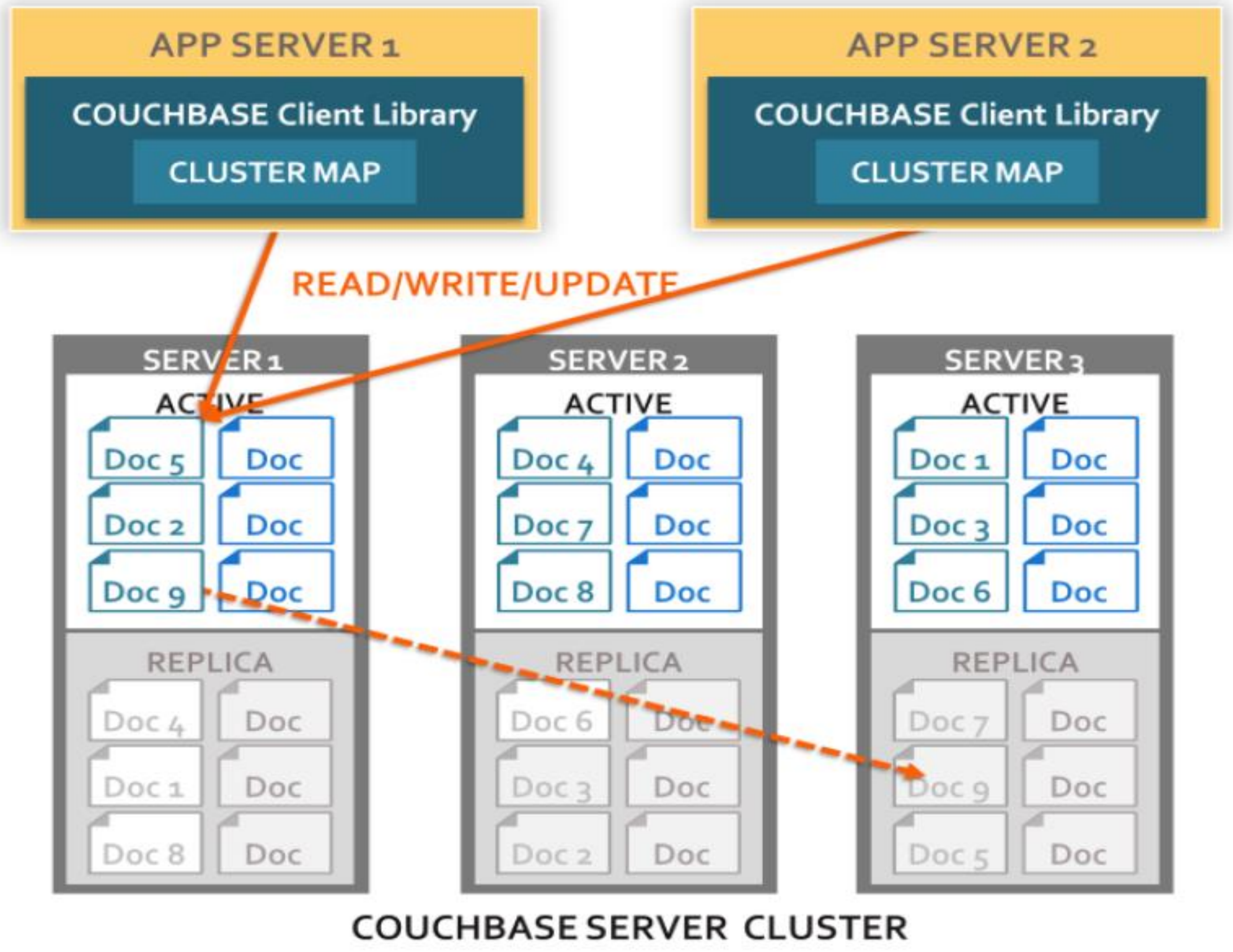
Easy to change data without database changes and downtime

Typical Couchbase Production Setup



Couchbase Operation

- Docs distributed evenly across servers via vbuckets+hash algorithm.
- Each server stores both active and replica docs in memory and on disk.
- Cleint library provides app with simple interface to database.
- Cluster map provides map to which server doc is on.
- App never needs to know
- App reads, writes, updates docs.
- CAS is an acronym for *Compare And Swap*, value to control how concurrent document modifications are handled



Cocuhbase concepts (Stuff that matters): Replication and Rebalancing

- **Intra-cluster** replication to handle node failure
  - 1 copy (Active + Replica) good for 1 node failure, “N” copies for simultaneous “N” node failures
- **Inter-cluster** replication (XDCR) for cluster failure (and more)
  - **Pros:** Handle cluster level availability – Upgrades, Traffic shifts, etc.
  - **Pros:** Enables > 1 node failure without additional replica copy
  - **Cons:** Additional infrastructure, load, connections and complexity for XDCR
  - **Cons:** Full XDCR refresh may be triggered in certain cases (quite heavy and impactful)
- Rebalance operations
  - Required vBucket redistribution post “cluster event” (Planned/Unplanned node failure)
  - Views and View replicas require special handling as Rebalance progresses
  - Need to manage consistent state as mutations occur even as “N” of 1024 vBuckets “moved”
  - Rebalance duration: # **of Worker threads**/Buckets/Data size/Views/Mutation rate/# of copies

Cocuhbase concepts (Stuff that matters): TTL and Compaction

- TTL – Time-To-Live
  - Set at record level (code controls this, not the DBA!)
  - Controls data expiration (and size/capacity of cluster indirectly)
- Compaction
  - Natural result of expiration/deletion/mutation – changes/tombstones appended to disk
  - Merging of multiple vBucket files removing “deadwood”, retaining active data
  - Write amplification – Original write + Rewrite at compaction
  - Heavy Write I/O and resource usage during compaction – Processes dedicated to compaction
  - Competes with application access – need to tune/time
  - Data files (vBuckets), Views and GSI undergo compaction – All representations churn!
  - Additional disk space required for storing mutations and during compaction
  - Percentage of fragmentation and timing of compaction set at cluster level (not bucket level)

Cocuhbase concepts (Stuff that matters): Views and GSI

- Views
  - Secondary key access in its simplest form
  - Transformed and persisted structures apart from K-V Data; ***Eventually consistent*** with K-V Data
  - Map-Reduce organized Javascript code allows great flexibility
  - Spatial Views for location aware applications
  - View auto-partition local to that node, ***Requires scatter-gather*** operation for access
  - Not accessed via Managed Cache, rather ***accessed via OS File buffers***
  - Programmatic access via View API
- GSI – Global Secondary Index
  - Built primarily for N1QL queries using Multi-Dimension Scaling (Index Service)
  - Efficient Singleton lookups as well as Range scans
  - Accessed via Managed Cache, hence more predictable performance (and options!)

Cocuhbase concepts (Stuff that matters): Statistics

- Admin Console and REST API/Command line accessible
  - Trend and Track over time : Minute/Hour/Day/Week/Month/Year
- Stats: Server Resources
  - High level indicator of OS/Server as well as some CB specific stats (8091 stats?)
  - Supplement with your own server side stats
- Stats: Ops/Sec (Broadly Gets + Sets + XDCR Ops)
  - Leading indicator of activity at bucket/cluster level
  - Use YCSB benchmark on your configuration to determine your rough numbers
- Stats: Summary Stats
  - ARR: Active Resident Ratio (Lower value means **"Server-side" cache miss** from RAM)
  - RAM Usage: Low/High Water marks and Memory used
  - Cache Miss Ratio – Should track inversely with ARR, but depends on %age of real cache misses

Cocuhbase concepts (Stuff that matters): Statistics

- Stats: Summary Stats (contd)
  - Number of Items : leading indicator of capacity used (RAM and Disk)
  - Disk size should normally track “Doc fragmentation %age”: Spikes on rapid mutation rate
- Stats: Views and Query (MDS)
  - Summary of Data size, fragmentation and read stats in Summary Stats
  - Size and read stats for individual design docs
  - MDS: Index usage and Query stats at Cluster level
- Stats: XDCR
  - Drain rate (items and bytes)
  - Items remaining
  - Incoming and Outbound stream stats

Cocuhbase concepts (Stuff that matters): Configuration Parameters

- MDS (Multi-Dimension Scaling) vs. Homogenous configuration
- Alerting and Monitoring: Out of Box (use all) plus Additional
  - OS/Host level alerts, Finer-grained usage alerts
- RAM allocation – Data/Index and View
  - Metadata - Value Ejection (default) vs Full Ejection (understand full implication before use!)
- Autofailover : ”Yes” is a no-brainer, but the Time setting is important
- Auto-compaction: Percentage and Timing
  - Metadata Purge Interval : Reduce to 1 day for Volatile buckets
- “Read Only” user is not what it seems to be (Config/Stats only)
  - RBAC for Admin in 4.6; More to come in 5.x and above
- Hidden settings available via REST API Only (e.g. number of compaction threads)

Cocuhbase concepts (Stuff that matters): Deployment options

- VM vs. Bare Metal - overlaid by On-Prem vs. Public Cloud
  - Cloud : Elasticity, PaaS vs IaaS model, Managed Services, Not on your premises
  - On-Prem : Options for Bare Metal, Close to Legacy, Private Cloud options
- Failure protection models
  - Single cluster – “N+1” copies for protecting simultaneous “N” node failures
  - Multi cluster – 1+1 only, failover to surviving cluster for larger events/traffic shifts
  - Overhead: Requires XDCR or custom built replication (write everywhere from app layer)
  - Complexity: App level topology awareness, server failure recognition and switchover
  - Data Quality: Increased chances of data conflict due to bi-directional replication
- SoR (System of Record) or Cache (System of Engagement)
  - Depends on data model complexity, failure/data loss tolerance
  - Need to build supporting tools (ETL, Ability to reconstruct, Traceability)
- Multi-Tenancy : Cluster and Bucket level

Data Modeling – Model the data for your needs

- Data is a critical piece of the system : “Applications come and go; Data stays Forever”
  - Data model creates efficiencies: Access, Data Quality, Scalability
  - NoSQL pushes model responsibilities (maintenance/evolution) to application layer
- Choice of Key: “Key is Key”
  - Rekeying is very difficult, but possible (and may be essential as you grow)
  - Differentiate your keys for multi-tenancy
  - Use 2- or 3- character prefix
  - Remember: All keys + metadata (56 bytes) WILL remain in Bucket RAM (default: Value ejection)
- Avoid Secondary key access if possible
  - Views and GSI is eventually consistent with K-V Puts – Immediate RYOW could be a problem
  - Additional complexity and overheads : MDS provisioning, Scatter-Gather
  - Pushes responsibility to Server-side : simplifies development in many cases

Data Modeling – Model the data for your needs

- Complete your Logical Data Modeling before Physical Data Modeling
  - Logical: Entitities, Attributes and Relationships
  - Understand Buckets, Keys, Items (documents)
  - Simple Key-Value or JSON Document
- Understand Relational to NoSQL mapping and prepare to modify/adapt

Relational Databases	Couchbase Server
Databases/Schema	Buckets
Tables	Documents with type designator attribute (Key Prefix)
Rows	Items (Key-Value : Opaque or Key-JSON – Document)
Columns	Attributes
Index	Views/Index
Materialized Views (Oracle)	Map/Reduce Javascript logic based Views

Couchbase is a good fit for

- **Application Characteristics – Data driven:**
  - 3rd party data aggregation or user defined structure (Social media).
  - Support for unlimited data growth (Viral apps).
  - Data with non-homogenous structure.
  - Need to quickly and often change data structure.

- **Application Characteristics – Performance driven:**
  - Low latency critical.
  - High throughput (ex. 200000 ops/sec).
  - Large number of users.
  - Unknown demand with sudden growth of users/data.
  - Predominantly direct document access via key.
  - Read / Mixed / Write heavy workloads.

***Don‘t Use for：Querying by value, multi-operation transactions, relationships between data.

Development Tips

- Choice of platform with Couchbase SDK
  - Java, Node.js, Python, C/C++, …
  - Couchbase Starter kits available on Couchbase.com
- Out of box access kits
  - Command line access: K-V client (cbc) and SQL client (cbq)
  - GUI: Browser based document access (K-V CRUD) and Query Workbench
- Client side resiliency
  - Build in client side logging and failure handling (Read failure vs Write failure)
  - Consider L1 Cache on client side (use case specific)
  - Keep client versions updated even as you update Server side!
- Provide Version specific QA clusters : handling mismatch between Prod and QA/Test
  - QA specific issue: Need bucket multi-tenancy to support all Production deployments

Operation Tips – Making Couchbase work

- Alerting and Monitoring
  - Use Out of Box alerting for important alerts and Admin Console for most metrics
  - Command line and REST API access to metrics
  - Custom alerts via scripts – Connection count, Disk Queue, CPU, Active Resident Ratio, etc.
  - Other monitoring frameworks (e.g. Nagios)
  - Consider integration to your org's SPOG (Single-Pane-Of-Glass) for monitoring
- Capacity management
  - Understanding, recording and reporting on various capacity vectors
  - Periodic review and capacity additions
- Security – Build in from the start
- Production operations – Virtuous loop of monitoring, responding and fixing
- Involve Couchbase resources (requires support contract)

Operation Tips {contd.}

- Automate Everything
  - Automation is the key to efficient operations (install/configure/failure recovery/upgrades)
  - Create Standard Operating Procedures for common events and keep adding to them
  - Build in Client level resiliency : Application level failover while one cluster has an event
- Operate in “DevOps” mode
  - DBAs: Understand Application side Data Model and Application characteristics
  - Dev: Provide Client side logging to enable tracing and troubleshooting
  - DBAs and Dev: Work together on moving Couchbase environments to the next level
- Vendor and Community relations
  - Keep updated on release notes, patches, bugs, features from Couchbase
  - Attend Couchbase conferences and network with other users

Training and Learning

Training is optional, (Ongoing) Learning is Essential!

- Formal/Informal Training
  - Instructor led from Couchbase (Virtual and Classroom)
  - Self paced from Couchbase.com
  - Onsite
  - Various paths (Dev/Mobile Dev/DBA)
  - Conferences and Meetups
- Learn from other's experience
  - Use cases organized around many vectors
  - Couchbase Blogs (a great source of information)

Putting it all together

- Couchbase is Easy – But introducing and operationalizing NoSQL can be difficult
- Requires new way of thinking and working
- “Forewarned is Forearmed”
- Q & A and Wrap up
- Connect with me on Linkindin - http://www.linkedin.com/in/Pradeep-tummala

General Use cases : (Token / Cookies)

- Large number (10’s or 100’s of millions) of small items
- Typically High rate of reads and writes as tokens/cookies are read and written
- Cookies: Server side storage of user/machine characteristics
  - Reduces application chattiness/network transfer
  - ***Re-creatable*** from Client side in case of data loss/cluster failure (with impact)
  - Enables longer history on server side
- Tokens: Long lived indicators for authentication/authorization
  - Useful for supporting “Keep Me Signed In” options
  - ***Re-creatable*** from Token Generator in case of data loss/cluster failure (with impact)
- Fairly longer TTL’s
- Use of XDCR for multi-cluster, eventually consistent replication
  - Consider number and configuration of clusters (one-> many, ring of 3 or 4)

General Use cases : (Cache)

- L2 Cache (Out of process) for Systems of Record
- Cache improves availability/scalability/head room for SoR through Read/Write offload
- Understanding and building cache consistency is critical
- Various types of Cache:
  - ***Read Buffering***: Offload Read requests from SoR
  - Infrequently changing / Frequently accessed / Result set caching of expensive operations
  - ***Write Buffering***: Offload Write requests (and usually read as well) from SoR
  - Transient data / Preserve Event state
  - Write back / Write aside on Cache Miss : ***Reconstruct from SoR***
  - Just-in-time pre-load is a good option
- Short TTL – Few minutes to Few hours (churn due to fragmentation!)

General Use cases : (Configuration/Rules engine)

- Repository of slowly changing rules or configurations
- Read Heavy, Write less/rarely
- Store forever – No TTL
- Client side optimizations
    - Build a “Rules service” with L1 (in process) cache
    - Service layer caching provides high availability and read offload
    - Caching enables serving client even when Couchbase is unavailable
    - Build in ”updater thread” to refresh L1 cache periodically or on demand
- Data Model : Handling multiple rules versions
    - Ability to provide forward and backward compatibility
    - Support for N-1, N and N+1 version with auditability

***Don‘t Use for : Querying by value, multi-operation transactions, relationships between data.