Architecture and Administration Basics

Workshop Day 2 – Indexing

Indexing in Couchbase Server

- Map/Reduce Views – Data Service
  - Powerful programmable indexer for complex reporting and indexing logic.
  - Full partition alignment and paired scalability with Data Service.
- Spatial Views – Data Service
  - Incremental R-tree indexing for powerful bounding-box queries
  - Full partition alignment and paired scalability with Data Service
- GSI – Index Service
  - New global indexing for low latency queries without compromising on mutation performance (insert/update/delete)
  - Independently partitioned and independently scalable indexes in Indexing Service
  - Memory Optimized Indexes (MOI), New Standard GSI / Plasma (New in 5.0) and Legacy GSI / Forrest DB (being deprecated in 5.0)

Indexing

- **Local** secondary indexes (Views)
  - Co-located with data
  - Higher write performance
  - Lower read performance: scatter-gather
  - Scaling bottleneck with a high number of indexes or data nodes

- **Global** secondary indexes (Standard GSI / MOI *New in 4.5* / Plasma * New in 5.0)
  - Independently scaled and partitioned
  - Isolated from Key-Value operations
  - Higher query performance
  - Async writes to a large number of global indexes

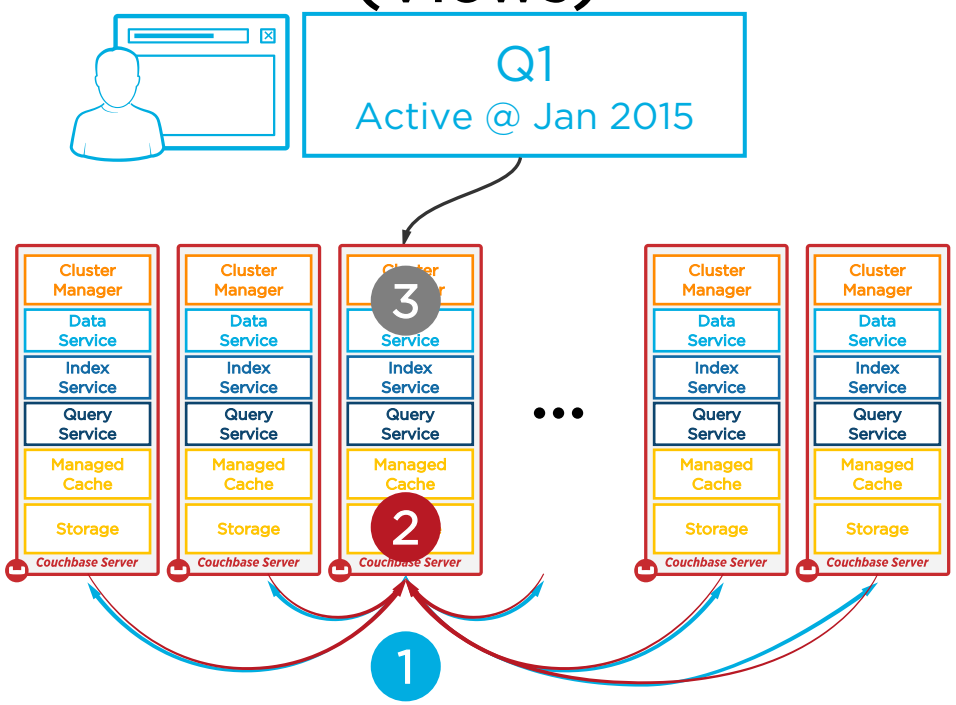
Indexing with Couchbase Server

Find the Top 10 Most Active Users for Month of Aug

```
INDEX ON Customer_bucket(customer_name, total_logins.jan_2015)

WHERE type="customer_profile";
```

```
SELECT customer_name, total_logins.jan_2015

FROM customer_bucket

WHERE type="customer_profile"

ORDER BY total_logins.jan_2015 DESC LIMIT 10;
```

Local Indexes (Views)



Q1: Execution Plan on N nodes

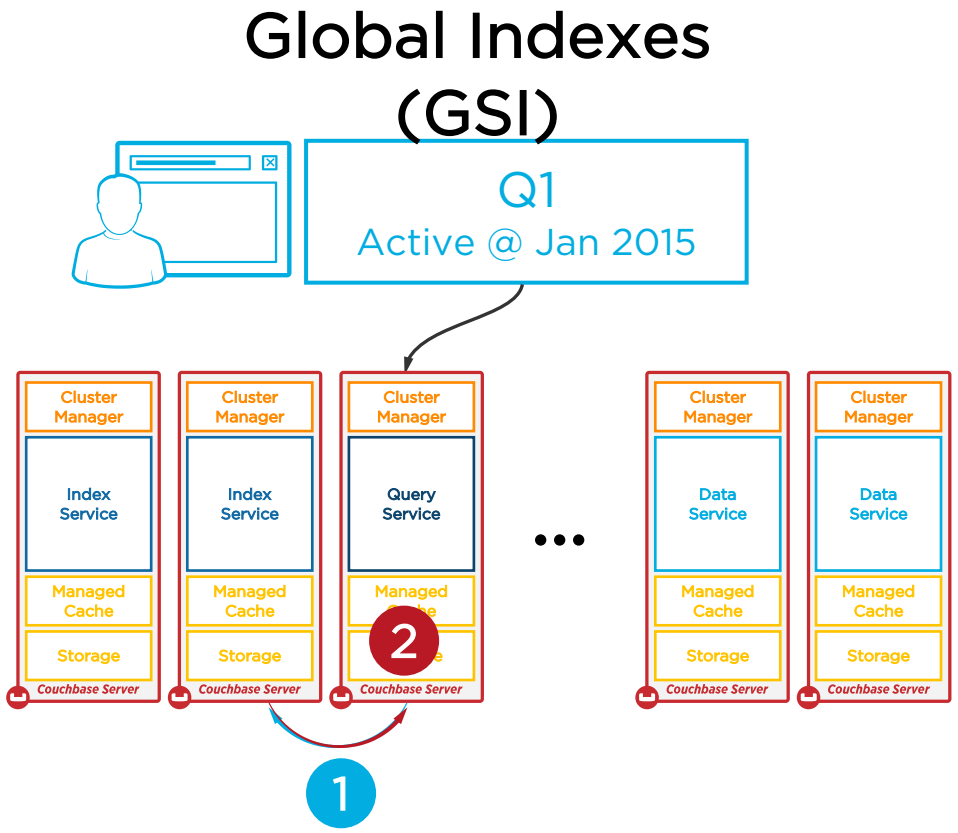
1. **Scatter**: Scatter Q1 to N nodes
2. **Gather**: Gather N results from N nodes
3. **Finalize**: Re-Aggregate Q1 on 1 node

Indexing with Couchbase Server

Find the Top 10 Most Active Users for Month of Aug

```
INDEX ON Customer_bucket(customer_name, total_logins.jan_2015)

WHERE type="customer_profile";
```

```
SELECT customer_name, total_logins.jan_2015

FROM customer_bucket

WHERE type="customer_profile"

ORDER BY total_logins.jan_2015 DESC LIMIT 10;
```



Q1: Execution Plan on N nodes

1. **Execute** Q1 on N1QL Service node
2. **Scan** index on Index Service node

Which to choose – GSI vs Views

Workloads	GSI	Map/Reduce Views
Complex Reporting	Just In Time Aggregation	Pre-aggregated
Ad-hoc Querying	Ad-hoc	View per Query
Flexible Indexing Logic	N1QL Functions	JavaScript
Secondary Lookups	Faster with Single Node Lookup	Slower due to Scatter-Gather
Range Scans	Faster with Single Node Lookup	Slower due to Scatter-Gather

Capabilities – GSI vs Views

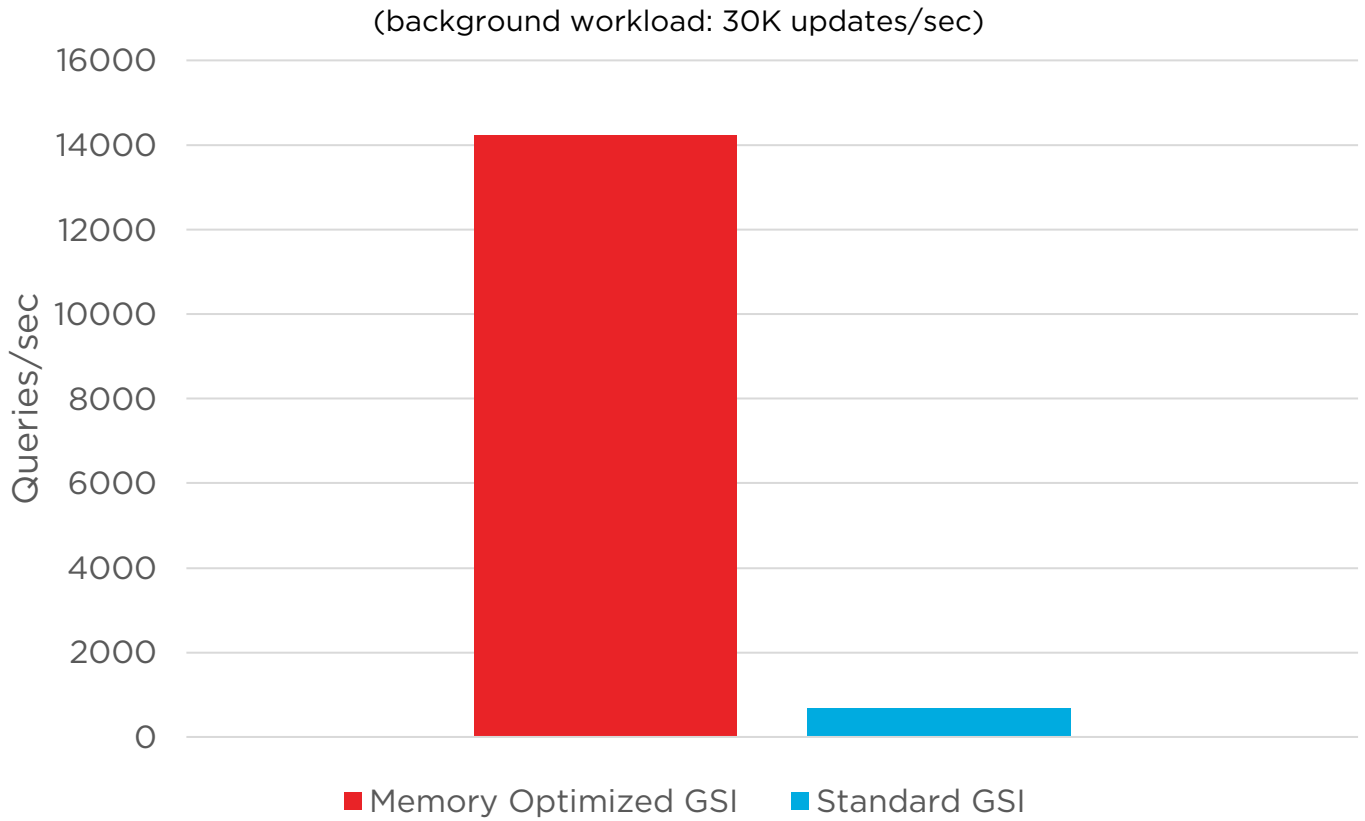
Capabilities	GSI	Map/Reduce Views
Partitioning Model	Independent – Indexing Service	Aligned to Data – Data Service
Scale Model	Independently Scale Index Service	Scale with Data Service
Secondary Lookup	Single Node	Scatter-Gather
Range Scan	Single Node	Scatter-Gather
Grouping, Aggregates	With N1QL	Built-in with Views API
Caching	Managed	Not Managed
Storage	Standard GSI / MOI	Couchstore
Availability	Manual with Multiple Identical Indexes	Auto Replica Based

Which to Choose – Legacy, Standard and Memory Optimized GSI

	Legacy Standard GSI (Forrest DB) [Being deprecated in 5.0]	New Standard GSI (Plasma)	Memory Optimized GSI
Initial Index Build Times	Slower	Significantly Faster than Legacy	Faster
Ongoing Index Build Times	Slower	Significantly Faster than Legacy	Faster
Ability to Keep up with Mutation Rate	Lower	Significantly Faster than Legacy	Higher
Memory Demand for Larger Indexes	Lower	20% Residency	Higher
Behavior on OOM (out of memory)	Spill to disk	Spill to disk	Stop processing mutations

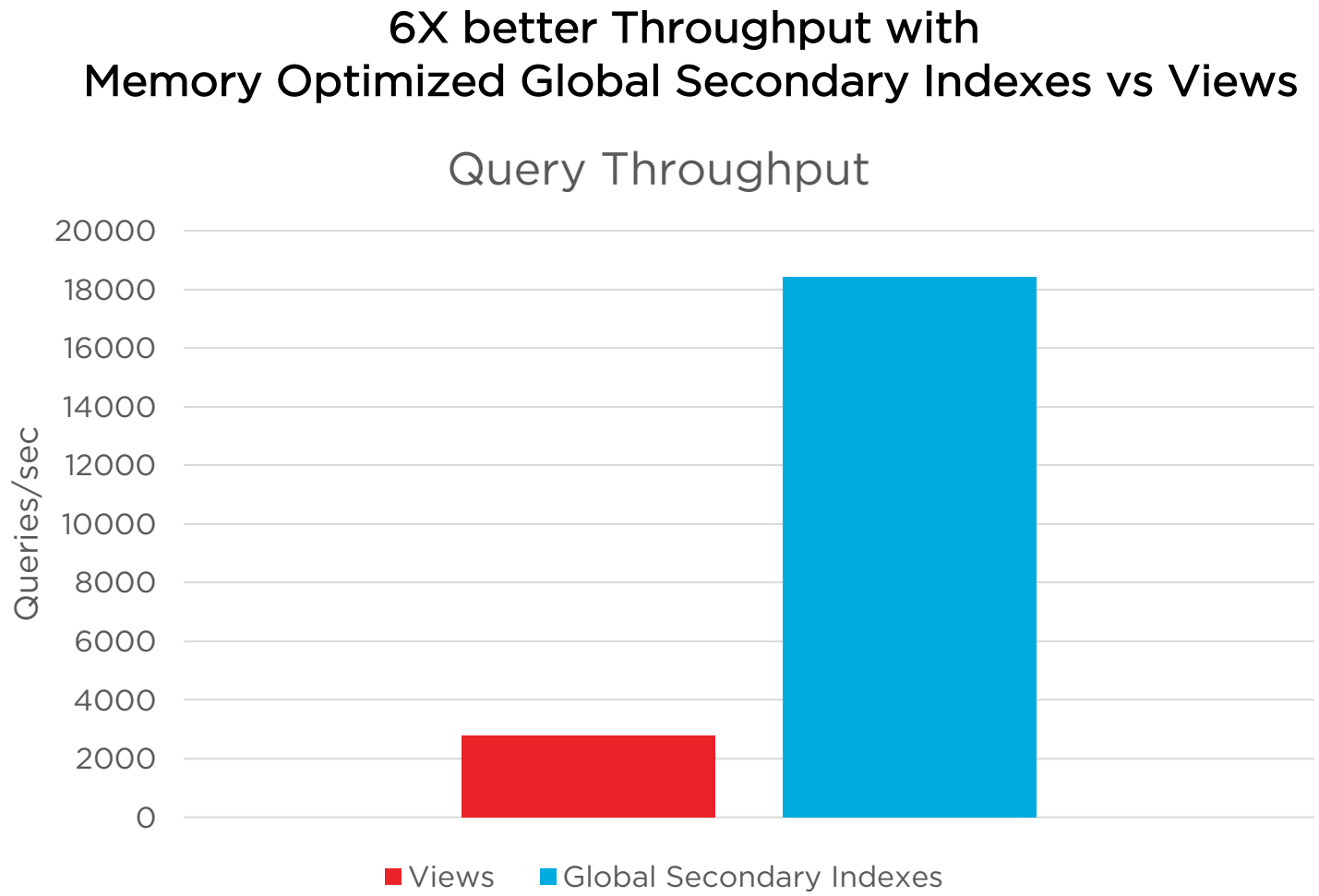
Faster Queries with Memory Optimized GSI

>20x Higher Query Throughput



Details: Query Throughput (queries/sec), N1QL 1 bucket 20M items with 1K size, Query: Singleton Unique Lookup with stale=false, Mutations: 30KSops, Index: MOI, HW: 6 nodes x 24 cores 128GB RAM – 4 data, 1 index and 1 query service node

Faster Queries with MO GSI over Views



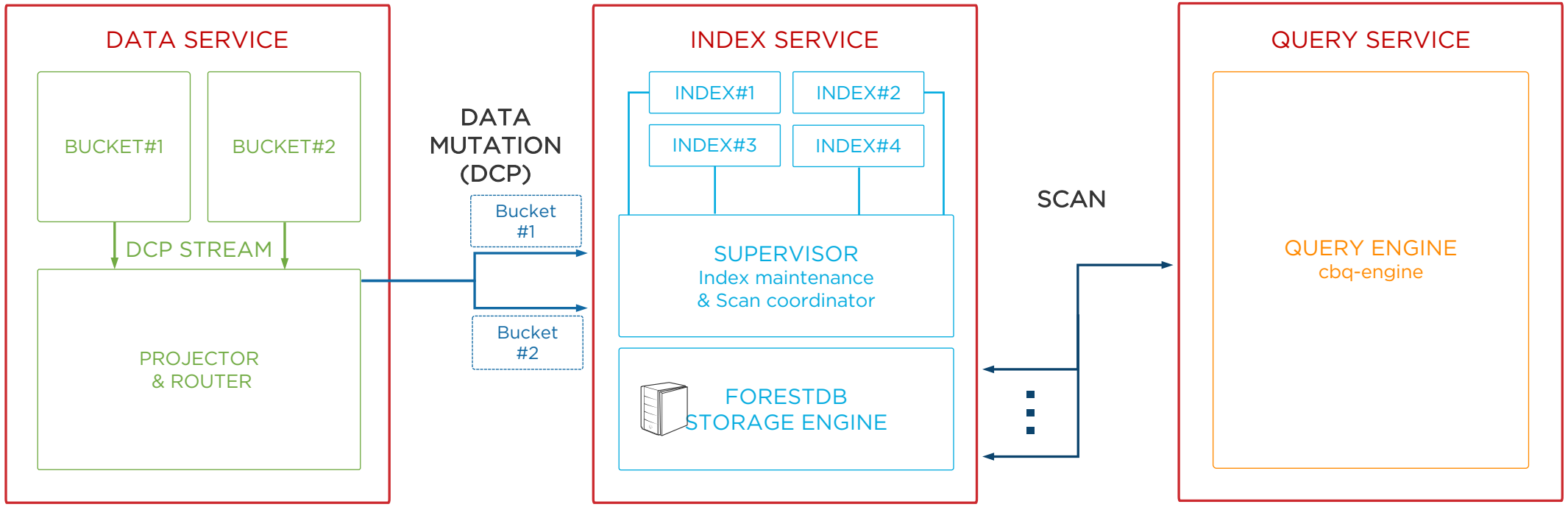
Details: Query Throughput (queries/sec), N1QL 1 bucket 20M items with 1K size, Query: Singleton Unique Lookup with stale=ok, Mutations: 2K/sec views 30K ops/sec for MOI - HW: 6 nodes x 24 cores 128GB RAM – 4 data, 1 index and 1 query service node

Couchbase

Global Secondary Indexes

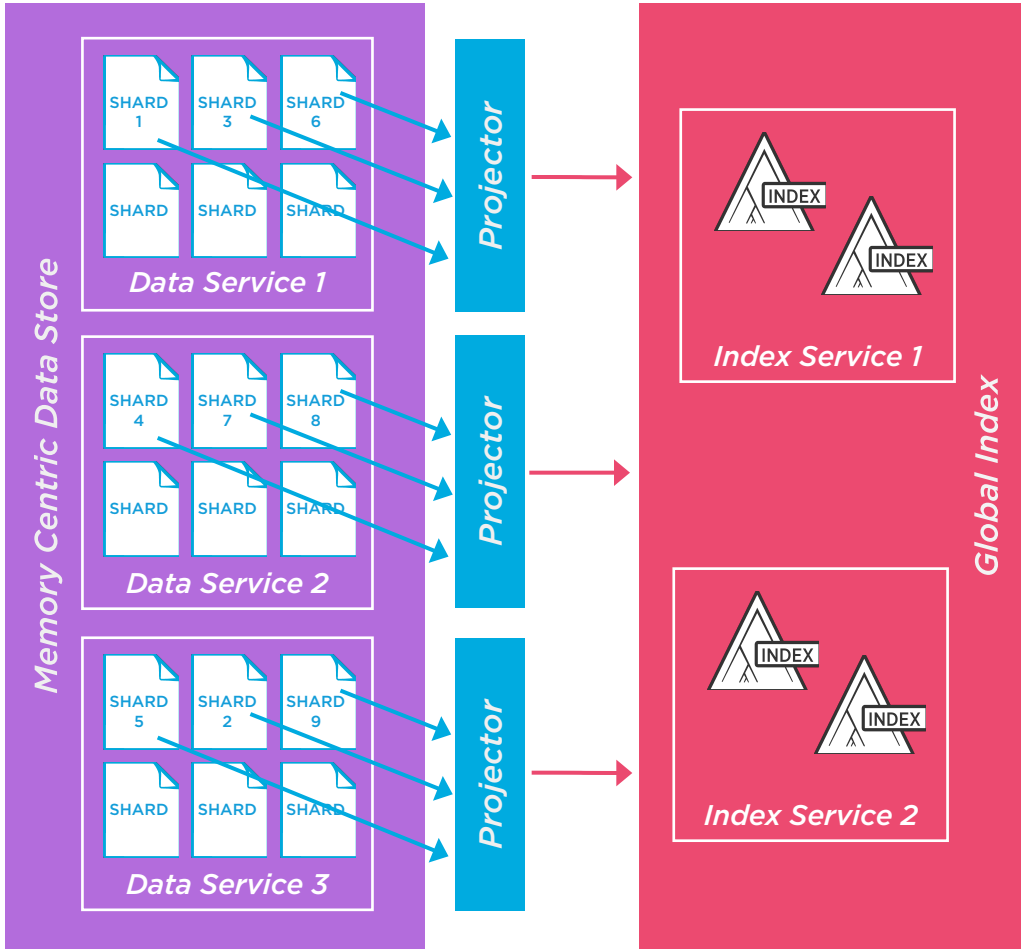
Under the hood

Couchbase 4.0 + Services



Global Indexes

- Indexing and scanning work is consolidated
- N1QL CREATE INDEX paradigm
- Indexer can process embedded functions and perform nested array traversal
- Each data service projects its own data
- Indexer can be pushed various scan types
- Index replicas are supported*

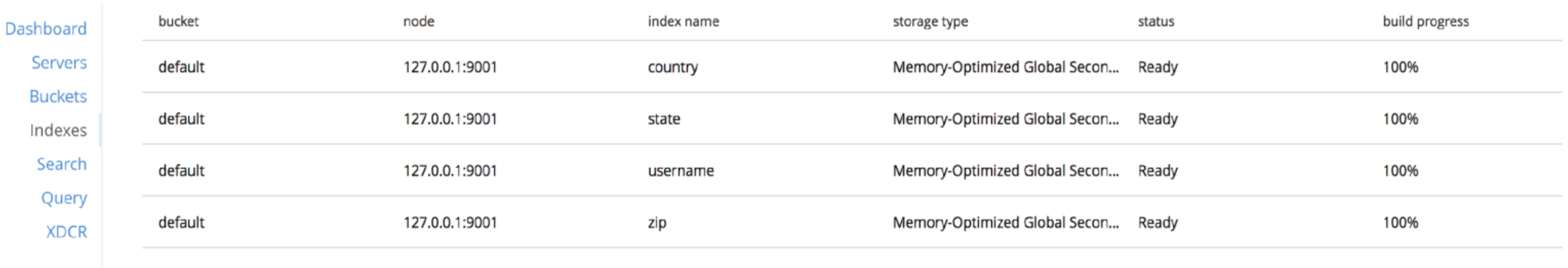


Step by Step – Building indexes

- Building and Index
  1. Issue “CREATE INDEX …”
  2. Place on the “next node” if WITH {“nodes”:[]}not specified
     - “next node” picked by round robin
  3. Create metadata for the index on the node
     - “next node” picked by round robin
  4. If deferred build = true
     - Mark Index Status = “Pending”
     - Wait for “BUILD INDEX …”
  5. Else
     - Mark Index Status = “Pending”
     - Start DCP Backfill with Projector
     - When Done – Mark Index Status = “Online”

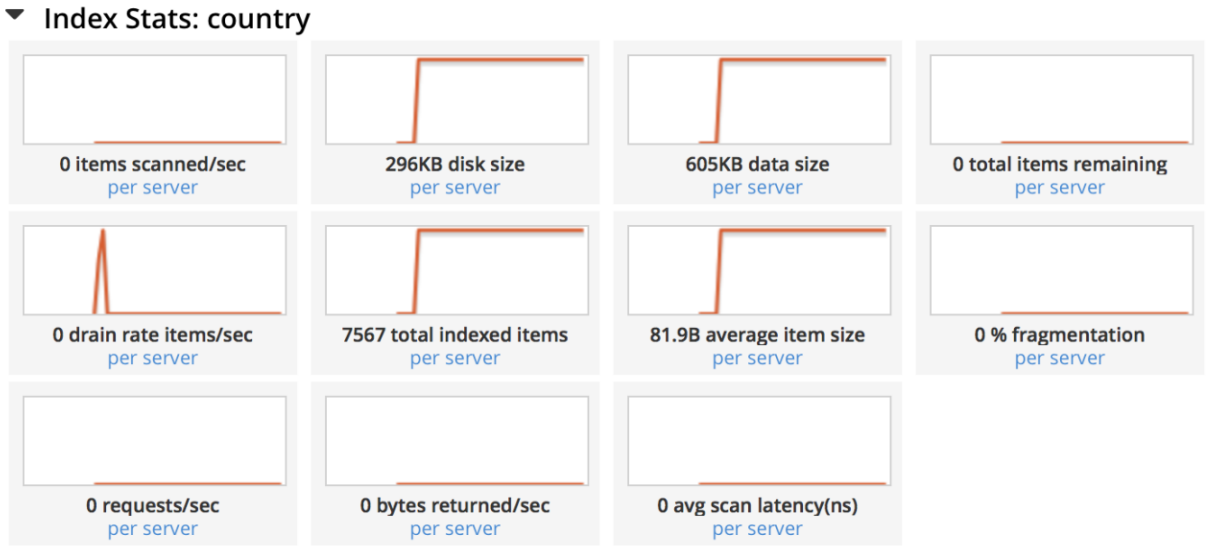
Index Status

- Show the status of each index
  - Created – Created but not yet built
  - Building – Index is being built
  - Ready – Index built is finished, ready for query



Index Statistics

- Statistics of each index is displayed under bu



Statistics

- Disk Size
  - Total bytes on disk (including fragmentation)
- Data Size
  - Total bytes used by index
- % Fragmentation
  - Standard index only
  - Percentage of obsolete data to be reclaimed
- Total Indexed Items
  - Total number secondary keys
- Average Item Size
  - Average size of secondary key

Statistics

- Total Items Remaining
  - Estimated number of pending documents remained to be processed
- Drain Rate items/sec
  - Average update throughput in terms of # of documents
- Request/sec
  - Scan throughput in terms of # of requests
- Bytes returned/sec
  - Scan throughput in terms of bytes
- Item Scanned/sec
  - Scan throughput in terms of # of returned keys/rows
- Avg Scan Latency
  - Average scan latency

Indexes & Partitioning

- Why Partition Indexes?
  - Index may not fit on the node
    - Fit index to the nodes you have
    - Distribute Index load to many nodes
  - Minimize the tree scanned
    - Large/Tall indexes will take longer to scan

Index Partitioning

- How to Partition Indexes?
  - By list of values – great for scan latency
    - `... WHERE type = “customer”`
    - `... WHERE type = “product”`
  - By range –
    - `... WHERE id BETWEEN 10 AND 20`
    - `... WHERE id BETWEEN 20 AND 30`
  - By function – no range scan ability
    - `… WHERE 0=id % 2`
    - `... WHERE 1=id % 2`

- Benefits:
  - Faster Build Time
  - Lower Maintenance Latency
  - Faster Scan Latency
  - Higher Scan Throughput

Step by Step – Ingest a Mutation

- Ingest a Mutation on Data
  1. DCP Feed communicates the mutation to Projector on the node.
  2. Projector picks up the mutation for a bucket
     - **Projector** strips the mutation to check relevance to existing indexes on each indexer
     - **Projector** Sends stripped mutation to indexers on all index service nodes
  3. Indexer receive the mutation on each index service node
     - Queue the mutation
     - **Extraction Worker** Prepares and queues the update to each index.
     - **Persistence Worker** persist the mutation to the index

Index Availability and Load Balancing

- Index Replicas (manual in version < 5.0) serve as equivalent indexes
  - Query Service will load balance as well automatically use the equivalent index if the original index is not available
  - It is recommended to keep replicas on different nodes and different availability zones

Index Service Deployment

- Recommended approach is to always have dedicated Index Service Nodes for production
- Co-deploy with Query/Data service only for small throughput workloads
- Co-deploying with Query is better, not recommended to deploy with Data service except in Dev/QA
- Standard GSI – Disk /Memory Intensive. Average CPU Utilization.
- Memory Optimized GSI – Memory/CPU/Disk Intensive

Couchbase

Indexing Options

Indexing Options

- Primary Index
- Secondary Index
- Array Index

Primary Index

- The primary index is the index on the document key on every document in a keyspace
- The primary index is used for:
  - Complete keyspace scan – equivalent of table scan
  - Query does not have any predicates (filters)
  - Query has predicate on document key
  - No other index or access path can be used
- Primary index is optional

```
CREATE PRIMARY INDEX ON `travel-sample`;


SELECT *
FROM `travel-sample`;

SELECT *
FROM `travel-sample`
WHERE META().id LIKE "user::%”;
```

- We *do not* recommend deploying a primary in index in a production environment

Secondary Index

- Simple Index
- Composite Index
- Functional Index
- Partial Index
- Replica Index
- Covering Index

Simple Index

- Index on any attribute or document-key
- Can use attribute within the document:
  - scalar, object, or array or expression
- Query WHERE clause must use index keys
  - Leading keys

```
CREATE INDEX ts_name ON `travel-sample`(name);
CREATE INDEX ts_city ON `travel-sample`(city);

SELECT *
FROM `travel-sample`
WHERE name = "United Airlines";

SELECT *
FROM `travel-sample`
WHERE city = "San Francisco";
```

Composite Index

- Index more than one expression
- Efficiently support queries with multiple predicates

```
CREATE INDEX ts_info ON `travel-sample` (type, id, name);

SELECT *
FROM `travel-sample`
WHERE type = "airline" AND id BETWEEN 0 AND 1000 AND name IS NOT NULL;

SELECT *
FROM `travel-sample`
WHERE type = "airline" AND id  < 100;

SELECT *
FROM `travel-sample`
WHERE type = "airline";
```

Functional Index

- Index any function or expression on the data, in addition to attributes
- Cannot contain aggregate expressions

```
CREATE INDEX ts_func ON `travel-sample`(UPPER(name), LENGTH(description));

SELECT *
FROM `travel-sample`
WHERE UPPER(name) = "MARRIOTT";

SELECT *
FROM `travel-sample`
WHERE UPPER(name) = "MARRIOTT" AND LENGTH(description) > 256;
```

Partial Index

- Index only a subset of the documents in a keyspace
- Create smaller, focused indexes
- Useful for keyspaces with multiple types of documents

```
CREATE INDEX ts_airline ON `travel-sample`(name) WHERE type="airline";
CREATE INDEX ts_hotels ON `travel-sample`(name) WHERE type = "hotel" AND country =
'United States';

SELECT *
FROM `travel-sample`
WHERE type = "airline" AND name = "United Airlines";

SELECT *
FROM `travel-sample`
WHERE type = "hotel" AND country = "United States” AND name LIKE "M%";
```

Partial Index

- Complex predicates in the WHERE clause of the index
  - Separate index for each state
  - Separate index for each year

```
CREATE INDEX ts_year ON `travel-sample`(orderid) WHERE SUBSTR(orderdate,0,4) = "2016";

SELECT *
FROM `travel-sample`
WHERE SUBSTR(orderdate,0,4) = "2016" AND orderid IS NOT NULL;
```

Partial Index

- Techniques
  - Partitioning into multiple indexes using ranges or hashing
  - Partitioning into multiple indexes onto distinct indexer nodes
  - Partitioning based on a list of values, e.g. an index for each state

```
CREATE INDEX ts_airline ON `travel-sample`(name, id, icao, iata) WHERE type="airline”
AND icao like "A%";

SELECT *
FROM `travel-sample`
WHERE type = "airline" AND icao like "A%" AND name = "American" ;
```

```
CREATE INDEX ts_year ON `travel-sample`(orderid) WHERE SUBSTR(orderdate,0,4) BETWEEN
2015 AND 2020;

SELECT *
FROM `travel-sample`
WHERE SUBSTR(orderdate,0,4) BETWEEN "2015" AND "2020" AND orderid IS NOT NULL;
```

Duplicate (Pre 5.0) / Replica (New in 5.0) Index

- Identical keyspace, index keys and index WHERE clauses
- Different names
- Query planner chooses one of the indexes
- Indexer can use chosen index or any identical index
- High availability, load balancing, scaling, throughput

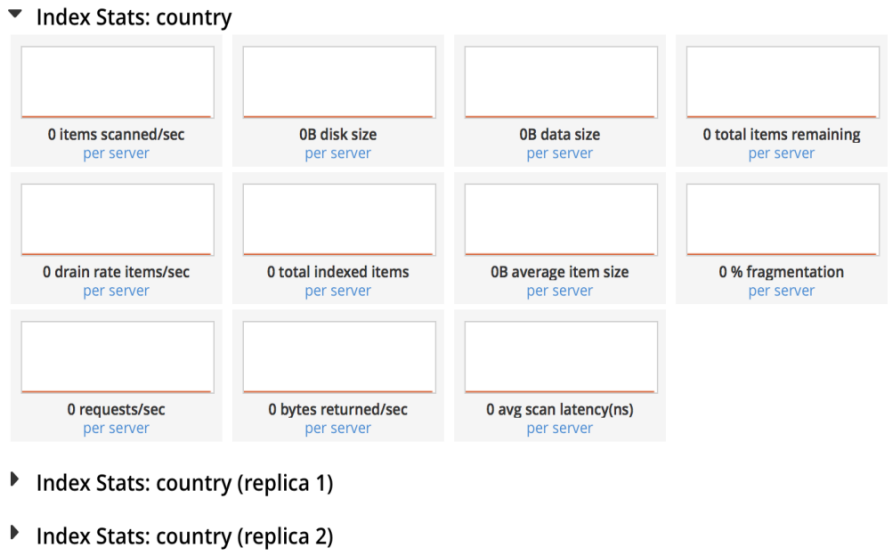
```
CREATE INDEX ts_di1 ON `travel-sample`(LOWER(name), id, icao) WHERE type = "airline";

CREATE INDEX ts_di2 ON `travel-sample`(LOWER(name), id, icao) WHERE type = "airline";

CREATE INDEX ts_di3 ON `travel-sample`(LOWER(name), id, icao) WHERE type = "airline";

SELECT name, id , icao, iata
FROM `travel-sample`
WHERE type = "airline" and LOWER(name) = "united airlines";
```

```
CREATE INDEX ts_di1 ON `travel-sample`(LOWER(name), id, icao) WHERE type = "airline” WITH
{“num_replica”: 3};
```

Replica Indexes: Auto Placement

Create index index1 on bucket(field1) with {"num_replica":2}

- Create 2 additional replica (total 3 copies)
- Couchbase picks index nodes for initial placement
  - Rule 1: Do not place replica onto the same node
  - Rule 2: Spread out replica onto as many server groups as possible
  - Rule 3: Pick the nodes that have fewest number of indexes (room for improvement in the future)
- As index is created, index build will run concurrently for each replica
- Index can be used for query as soon as any replica becomes online



Replica Indexes: Manual Placement

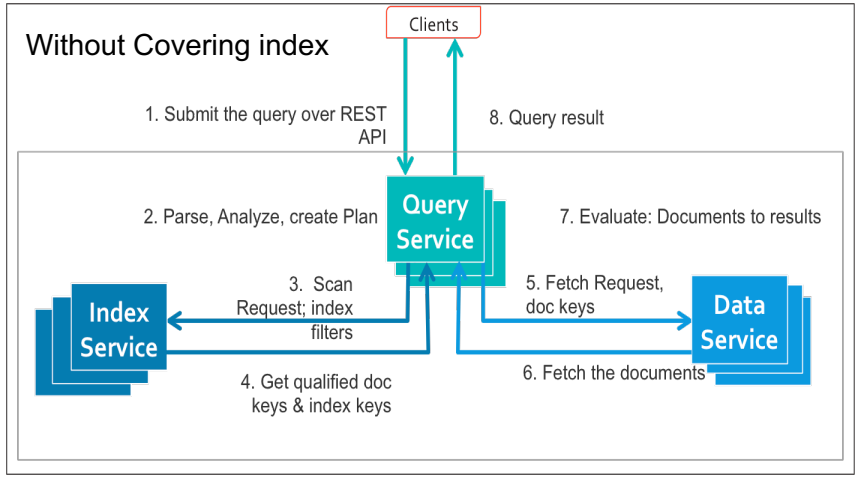
Create index index1 on bucket(field1) with {“nodes”:[“17.2.33.101:8091”, “127.2.33.102:8091”, “127.2.33.103:8091”]}

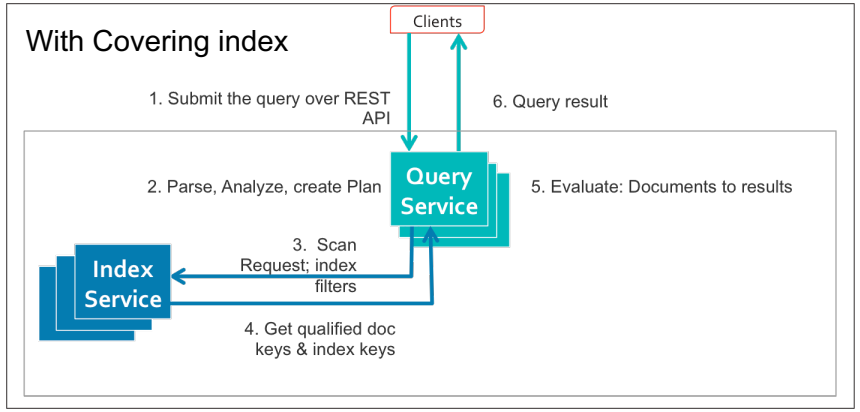
- Create 2 additional replica (total 3 copies)
- Administrator pick index nodes for initial placement
- Will override replica placement rules during initial placement

Covering Index

- Index selection for a query solely depends on the query predicates
- Index keys cover predicates and all attribute references
- Avoids fetching the whole document
- Performance

```
CREATE INDEX ts_airline ON `travel-sample`(name, id)
WHERE type = "airline";

SELECT name, id
FROM `travel-sample`
WHERE type = "airline" AND name = "United Airlines”;
```





Array Index

- Index individual elements of an array
- Stored array or computed array (functional)
- Supports ANY, ANY AND EVERY, UNNEST queries

```
CREATE INDEX ts_sched ON `travel-sample`
         (DISTINCT ARRAY v.day FOR v IN schedule END);

SELECT *
FROM `travel-sample`
WHERE ANY v IN schedule SATISFIES v.day <= 2 END;

SELECT *
FROM `travel-sample`
WHERE ANY v IN schedule SATISFIES v.day IN [0,1] END;
```

Array Index

```
CREATE INDEX ts_sched ON `travel-sample`

         (DISTINCT ARRAY v.day FOR v IN schedule END);



SELECT *

FROM `travel-sample`

WHERE ANY AND EVERY v IN schedule SATISFIES v.day <= 2 END;



SELECT *

FROM `travel-sample`
```

Array Index

- UNNEST using array indexing

```
CREATE INDEX ts_sched ON `travel-sample`
         (DISTINCT ARRAY v.day FOR v IN schedule END);

SELECT *
FROM `travel-sample` UNNEST schedule AS v
WHERE v.day <= 2;
```

Couchbase

Index Rebalancing

New in 5.0

Why do you need rebalancing?

- Add capacity for congested nodes or capacity planning
- Add capacity for new index
- Reclaim excess capacity
- Node Maintenance

Rebalancing Rules

- Redistribute indexes from ejected nodes to available nodes in cluster
- Will not redistribute index residing on non-ejected nodes
  - Preserve original layout as much as possible
  - Minimize index movement
- Preserve existing index layout if possible
  - When # of ejected nodes == # of new nodes
- Balance memory and CPU utilization if original layout cannot be preserved
  - When # of ejected nodes != # of new nodes

Rebalancing Replica

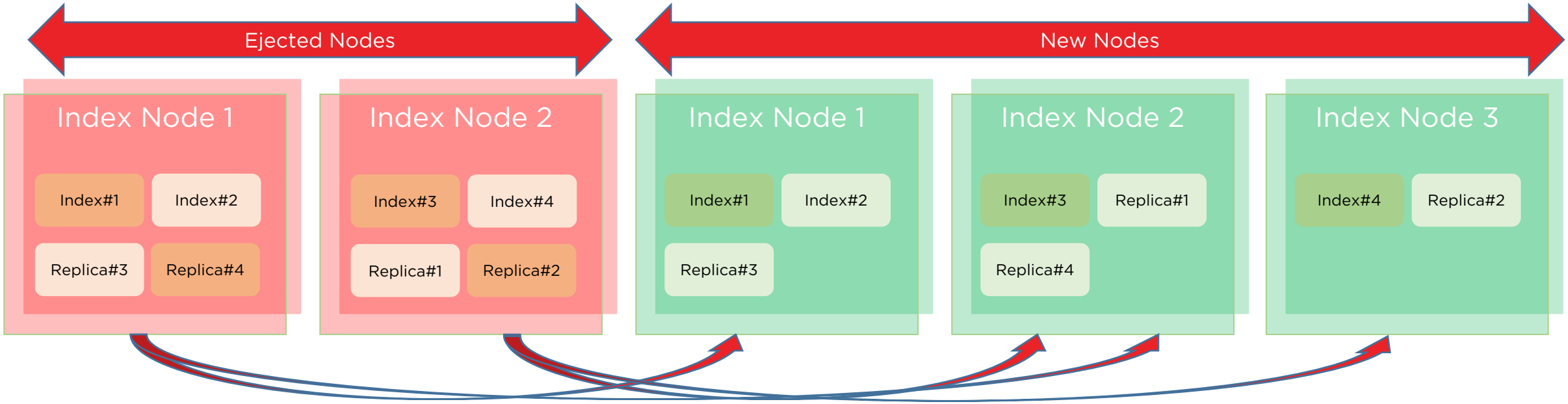
- Rebalancing will ensure the following
  - Do not place replica onto the same node
  - Spread out replica onto as many server groups as possible
- If there are more replica than available nodes, rebalancing can drop replica
  - Can repair lost replica when new node is added in next rebalance
- Rebalancing will not move indexes on a failed node
  - Can repair lost replica (in failed node) when there is sufficient capacity in cluster

Terminology

- Ejected Node: Node that goes out of the cluster after rebalance
- New Node: Node added to the cluster during rebalance
- Remaining Node (Non-ejected node): Nodes that remain in cluster after rebalance
- Available Node: Remaining Node + New Node

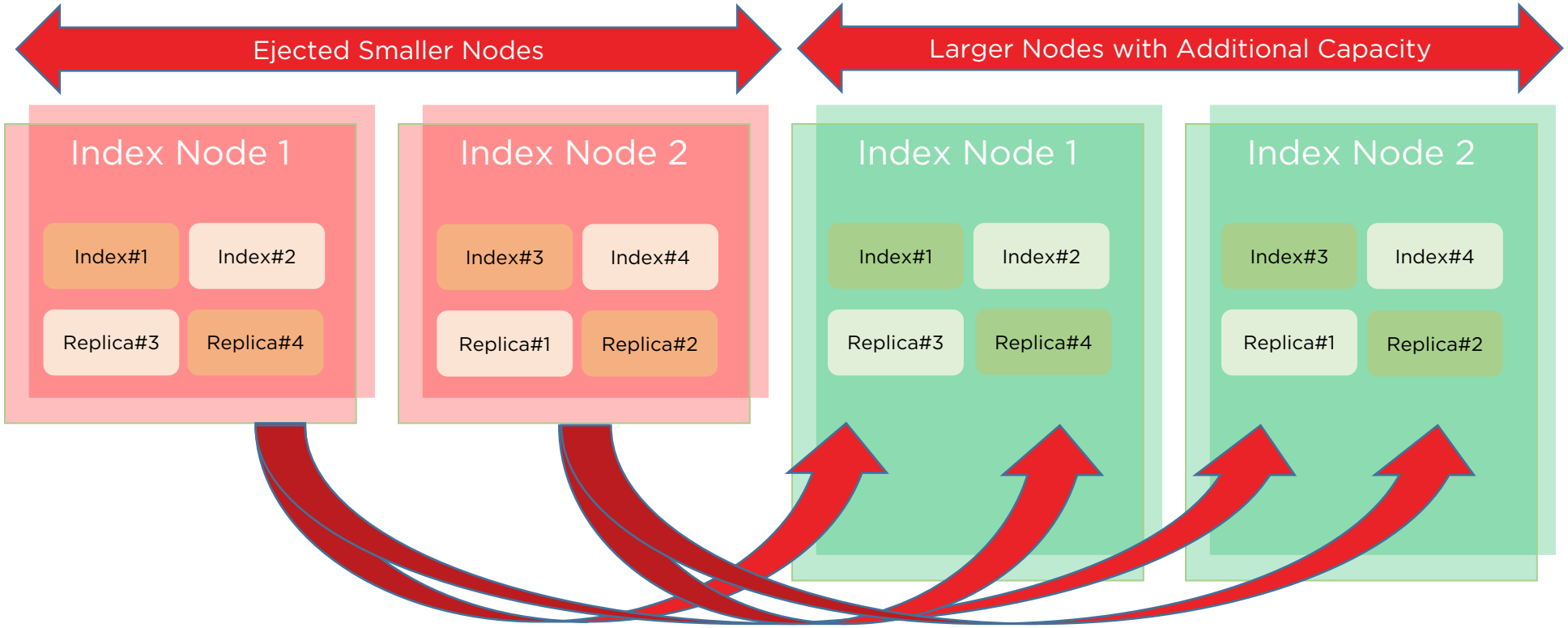
Add capacity for congested nodes or capacity planning (scale-out)

- Adding new capacity to relieve congested node (scale-out)
  - There are 2 indexer nodes under heavy load. Want to replace with 3 nodes to spread out the load.
  - Solution : Add 3 new nodes and eject 2 congested nodes
  - Will move indexes from ejected nodes to available nodes while balancing resource utilization



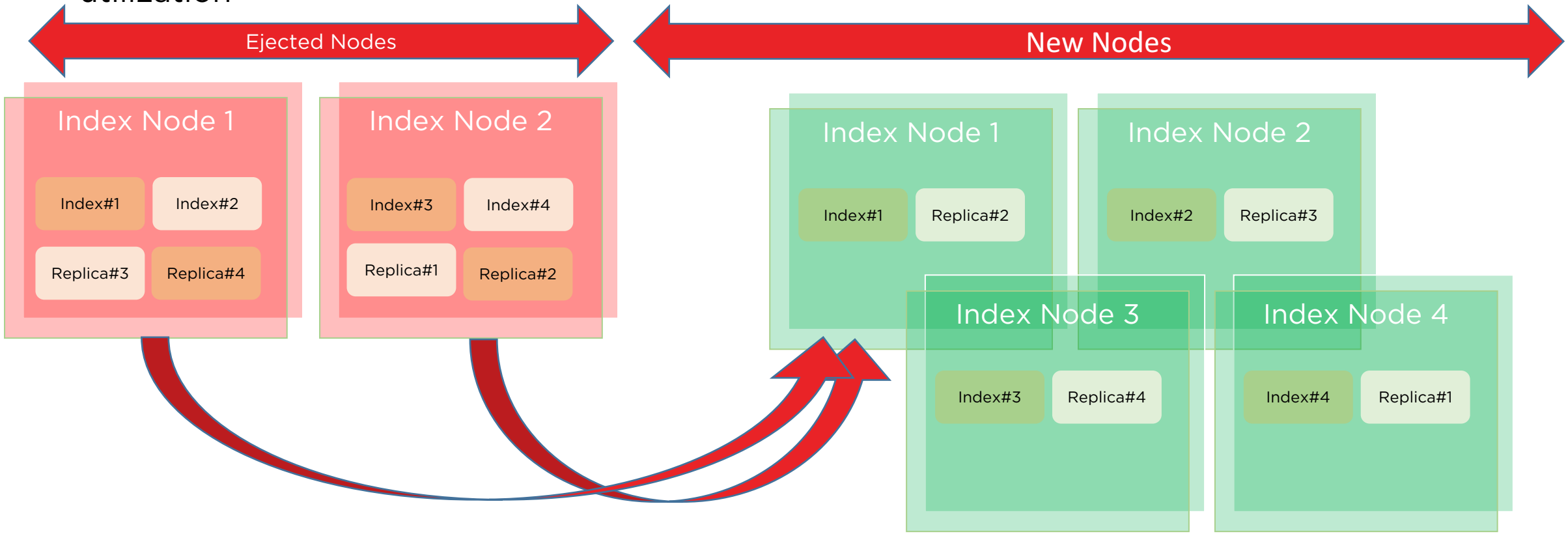
Add capacity for congested nodes or capacity planning (scale-up)

- Adding new capacity to to relieve congested node (scale-up)
  - There are 2 indexer nodes are under heavy load. Want to replace with 2 larger nodes.
  - Solution : Add 2 new nodes and eject 2 congested nodes
  - Will move indexes from 2 ejected nodes to new nodes while preserving index layout



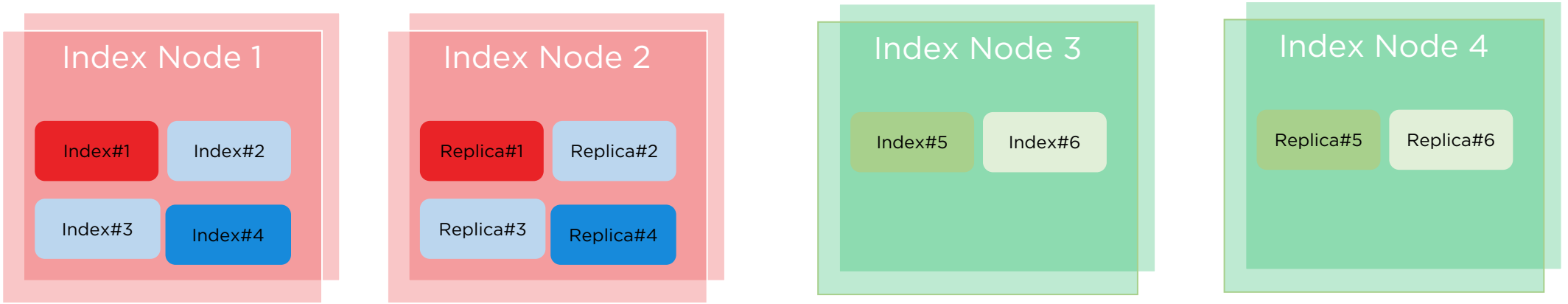
Add capacity for congested nodes or capacity planning (new capacity)

- Adding new capacity for resource planning
  - Need to increase capacity for 2 nodes for upcoming peak season. Want to add 2 nodes.
  - Solution : Add 4 new nodes and eject 2 nodes
  - Will move indexes from ejected nodes to available nodes while balancing resource utilization



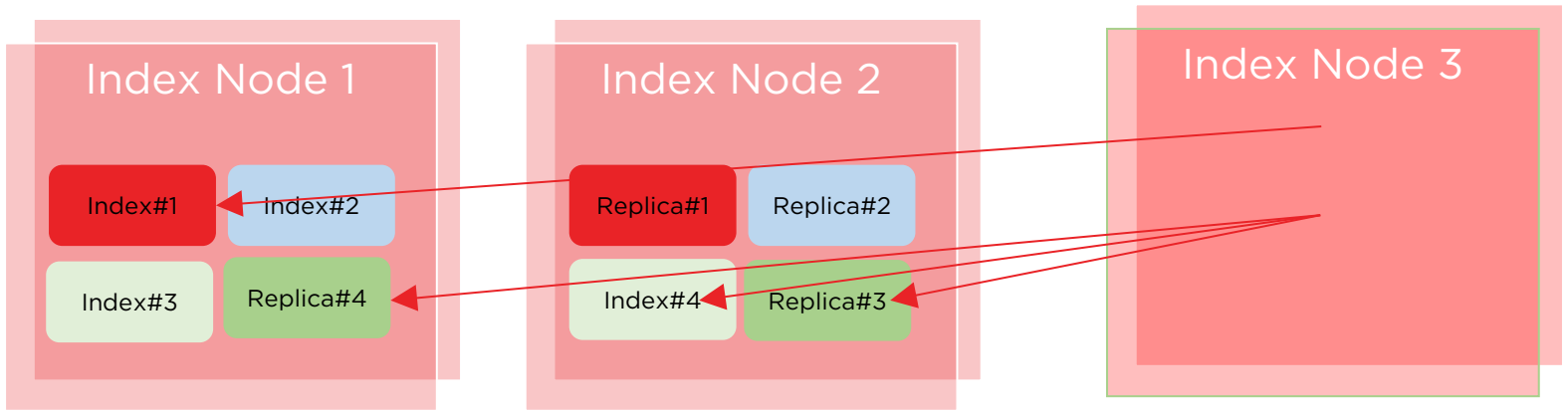
Add capacity for new index

- Adding new indexer node for additional indexes to be created
  - Need to add 2 more indexer nodes for additional indexes.
  - Solution : Add 2 new nodes and do not eject any node
  - Will not place existing index onto new node if there is no ejected node



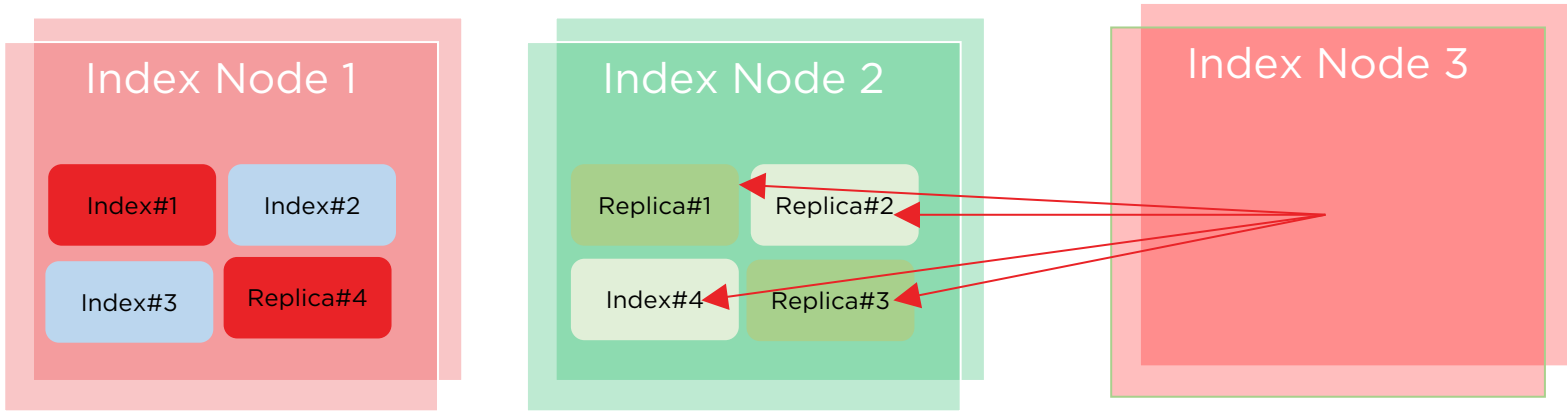
Reclaim excess capacity

- Reduce capacity (scale-in)
  - There are 3 indexer nodes. Want to reduce to 2 nodes when peak season is over.
  - Solution : Eject 1 node
  - Will move indexes from ejected nodes to available nodes while balancing resource utilization
- Reduce capacity (scale-down)
  - There are 2 indexer nodes. Want to replace with 2 smaller nodes when peak season is over.
  - Solution : Add 2 new nodes and eject 2 nodes
  - Will move indexes from 2 ejected nodes to new nodes while preserving index layout



Node Maintenance

- Node Maintenance
  - Need to swap out 1 node for maintenance
  - Solution : Add 1 new node and eject 1 node
  - Will move indexes from ejected node to new node while preserving index layout



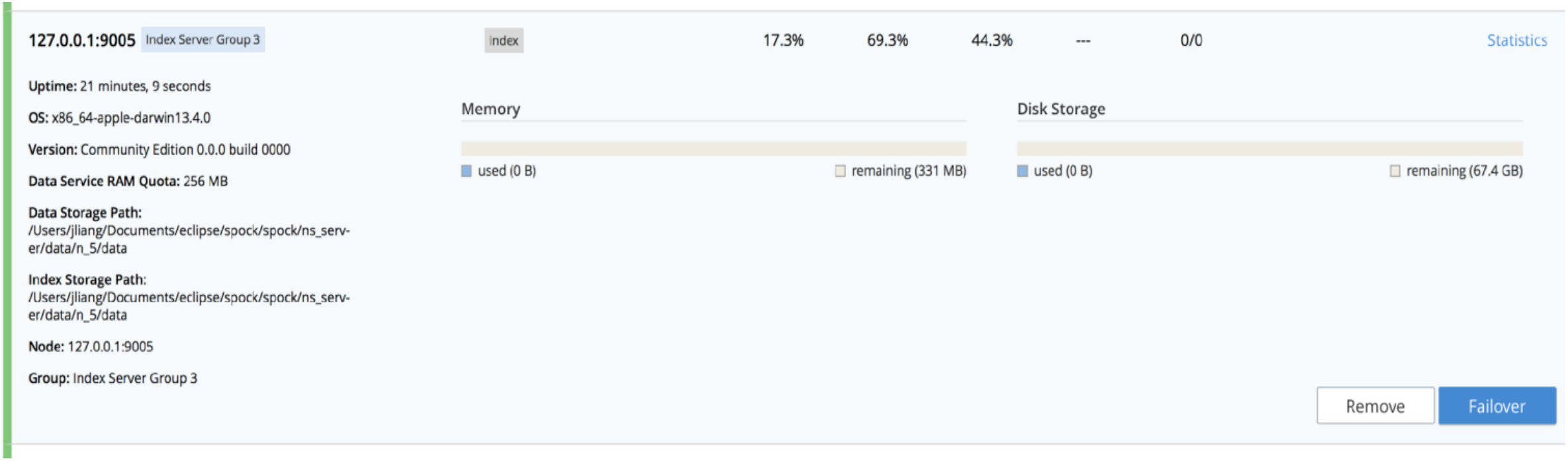
Couchbase

Failover / Delta Recovery

Failover and Delta Recovery

- Use Index Replica to ensure index remains online for query
- When index node recovers, it will recover both metadata and index data
  - Index will catch up to latest mutations
  - If an index has been dropped when index node is offline, it will be dropped upon recovery
  - If index build is executed when index node is offline, index build will start upon recovery

Failover and Delta Recovery



Couchbase

Backup / Restore

Backup / Restore

- cbbackup/cbrestore and cbbackupmgr
- Only Index Metadata is backed up
- Restore to the same cluster topology will result in same index layout
- Restore to different cluster topology will result in a different index layout

Monitoring - Stats

- Indexer collects a lot of stats related to its progress, performance of storage, scan timing, various other important operational aspects.
- Indexer Stats are published at runtime on REST endpoint

  http://localhost:9102/stats

- Indexer Stats are periodically logged at INFO log level as a JSON.