Secondary Indexes

Contents

Introduction

Goals

Functional Goals

Performance Goals

Key Assumptions

Architecture

Design

Projector Component

Responsibility

Collaborations

Router Component

Responsibility

Collaboration

Indexer

Component

Responsibility

Collaboration

Open Issues

Acknowledgements

# Introduction

A secondary index is a structure to accelerate lookups of documents by attributes besides the primary key. Secondary keys need not be unique. Some secondary indexes may offer iteration through secondary keys in collation order.

In general, a secondary index is a performance optimization feature and should not independently introduce other functionality that is not necessary for implementing the secondary index itself. This decision is to allow the feature to have clear goals so appropriate tradeoffs can be made to meet them.

# Goals

## Functional Goals

The following are the main functional goals for the project:

1. The index infrastructure should be able to build indexes and maintain them using UPR subscriptions.
2. The secondary index framework must be capable of handling a variety of indexing structures, such as B-Trees, Bloom Filters, Perfect Hashes etc.

## Performance Goals

1. When all components of the secondary index are run on a single commodity node with SSD, in 150% DGM scenario, the node should capable of achieving 25,000 operations per second per node.

These operations can be a balanced mixture of inserts, reads, writes and deletes. When the workload is skewed towards inserts, writes, deletes, the total performance should stay the same or increase.

1. To the extent possible, performance should scale linearly with number of nodes dedicated for indexing, both for reads and writes.

However, it assumed that the number of nodes dedicated to index is significantly smaller than nodes dedicated for data storage, and the size of attributes indexes is a small slice of the full document.

## Key Assumptions

1. Number of nodes dedicated for indexing is much smaller than total number of nodes in the system holding the actual documents
2. Secondary indexes cover only a small slice of the full document, and cardinality of the index does not exceed cardinality of the bucket.
3. When a tradeoff must be made, write throughput (insert, update, delete) is more important than read throughput.
4. Choosing and composing indexes is the role of the query planner and optimizer and not directly functionality of the index itself.

# Architecture

# Design

## Projector

### Responsibility

The projector is responsible for taking the document stream and extracting relevant attributes of interest, evaluating expressions and pushing results to the router.

### Collaborations

This component collaborates with:

* Router: The projector pushes the results to router to stream it to the indexer nodes. It may also obtain its state by querying indexer nodes via the router.

* UPR: The node needs to talk to KV storage via UPR to obtain the initial set of documents to build the index, and the mutation stream to keep it up-to-date.
* Cluster Manager: To know which UPR streams to subscribe to from a given data node in the cluster.
* Index Manager: To know index definitions and routers

## Decisions

1. Projector will use the largest applicable subset of Query Engine expression language and share the expression evaluation module from the Query Engine.
2. The projector component will run on the KV node in the initial version (deployment choice), but will be functionally capable of running on any node.
3. The output of projector node will not use UPR to transport its output to router component. An internal protocol will be used.
4. Sockets will be used for transport to router. When collocated and on Unix platforms, Unix Domain Sockets will be used.

## Router

### Responsibility

The router is responsible for directing the output of projector to the indexer nodes. A simple router could just broadcast every entry to every index node. A complex router may track indexer node state and minimize the update messages on the network.

### Collaboration

The router will interact with:

* Projector: To source of indexing data
* Indexer nodes: The targets of the indexing data
* Index Manger: To know index definitions, partitions and indexer nodes

## Decisions

1. The router will not maintain the reverse index in the initial version, and so it will not be aware of old value locations. This is because we want to minimize the number of components that maintain a state parallel to KV.
2. When an update occurs, the router will broadcast a short message containing sufficient information to allow index nodes to drop stale data to all index nodes
3. The projector component will run on the KV node in the initial version (deployment choice), but will be functionally capable of running on any node.
4. The output of projector node will not use UPR to transport its output to router component. An internal protocol will be used.
5. Sockets will be used for transport to router. When co-located and on Unix platforms, Unix Domain Sockets will be used.

## Indexer

### Responsibility

The indexer is responsible for:

1. Storing secondary keys and associating it with documents in a fast and efficient manner
2. Answering queries for secondary keys and responding with access path to documents
3. Maintaining state relationship between KV data and indexed data, so that queries have a deterministic behavior.
4. Implementing ordering of keys. Initially, it will implement N1QL specific ordering of secondary keys.
5. Handling large cardinality duplicates efficiently. For example, the user should be able to create secondary indexes on Boolean expressions and still get some performance benefit from the index.
6. Efficiently purge stale data out of query path.

### Collaboration

The indexer will interact with:

* Query Engine to create and drop indexes
* Query Engine to speed up access paths needed by Query
* Router to obtain data stream and provide indexing state data to router (which will in turn deliver it to projector nodes)
* Index Manager to discover other nodes and index partitioning

## Index Manager

### Responsibility

The index manager is responsible for maintaining a list of index definitions, index node discovery, keeping track of index partition maps and repartitioning indexes.

### Collaboration

The index manager will interact with all components of the system.

# Open Issues

1. Should index manager be folded into cluster manager?
2. How do we describe index deployment plans? Will the management container proposal be implemented in some form?
3. Do we need to scale reverse index (stale data purge) independently of the indexer nodes? Need data.

# Acknowledgements

This document does not introduce any original ideas. It subsets ideas, assigns values to parameters and generally derives from below works.

[Distributed Indexing Design Proposal – Steve Yen](https://docs.google.com/document/d/1TEY_yjUMs3FT3FZkgqIZUiKziUUBGiCWFSjWOOj65Dw/edit?pli=1)

A proposal of an indexing architecture that can accommodate various index types we have in one framework.

[Consistency in Indexes – John Liang](https://docs.google.com/document/d/1VespzgCKgPLFwCGRrx0VW7RC_PkSXh6cU7FSaA9gjic/edit?usp=sharing&pli=1)

A proposal on how indexing can offer consistency attributes with reasonable performance and memory overheads.

[Couchbase View Indexes – Filipe Manana, Damien Katz](http://docs.couchbase.com/couchbase-manual-2.0/#views-and-indexes)

The existing materialized views sets a lot of the foundation on what indexes should look like in Couchbase, consistency and other functional aspects.

[Distributed Range Partitioned Indexes – Steve Yen, Dustin Stallings](https://docs.google.com/presentation/d/161vtnjDFOpnliT0DA2k1-jCRKiB-eY9JZH%20hd0YQoAXo/edit#slide=id.g62174401_0_28)

Architectural proposal on how range partitioned indexes should be organized and scaled in Couchbase.

[Management Container – Steve Yen](https://docs.google.com/a/melkote.com/document/d/1qZQE8pCgpcscmNDy9jpGV5g1f0U3t5cDK28NDolW_8U/edit?disco=AAAAAGh6iDQ)

A proposal on how the cluster manager could manage different nodes serving different purposes like indexer, KV etc.

[Durable Transactions – John Liang](https://docs.google.com/document/d/11Rm2fW9Punx4tktUkK5Re9P87N8_aUwaEJw3u7XR_UY/edit?usp=sharing&pli=1)

A proposal on building durable transactions in Couchbase.

[Index Consistency and Isolation – Steve Yen](https://docs.google.com/document/d/1Y_aXMUBzEvLf8PO8CJYv5eYiQmKsNYzMr6Fq30Cl6xg/edit?pli=1)

A discussion on consistency in indexes

[UPR Requirements – Steve Yen](https://docs.google.com/document/d/1pGgQCdHocyfdG61skkIeSfantCdCVmZV8hJf92ADwtk/edit?pli=1)

A proposal of UPR requirements for indexing and other tasks