eveloper driven approach to building secondary inde

Garren Smith - garren@tigrisdata.com

Agenda

1. Index Design And FDB
2. Index Building With A Queue
3. Managing Schema Upgrades
4. Fixing Atomic Hotspots

About Me

1. CouchDB PMC and 🥭
2. Prisma ORM
3. 🧗 📸 🇿🇦

Who Is Tigris Data

- A modern fully open-source Data Platform
- Document Database built on top of FDB
- MongoDB compatibility
- Semi-structured collections
- Full Text Search indexes

1. Index Design And FDB

Tigris Schema

```
@TigrisCollection("user")
export class User {
  @PrimaryKey(TigrisDataTypes.UUID, { order: 1, autoGenerate: tru
  id?: string;

  @Field({index: true})
  name: string;
  @Field({index: true})
  age: number;

  @Field(TigrisDataTypes.OBJECT, { index: true, elements: Locatio
  location: Location;

  @Field(TigrisDataTypes.Array, {index: true, elements: string })
  interests: Array<string>;
}
```

Document

```
{
    id: "<UUID>",
    name: "garren",
    age: 20,
    location: {
        country: "RSA",
        home: {
           address: "25 Sunny Road"
        }
    },
    interests: ["rugby", "music", "climbing"]
}
```

FDB Key Format

```
[idxs, ...key_path, version, value, dup_key, doc_id]
```

Storing In FDB

```
[idxs, ...key_path, version, value, dup_key, doc_id]
[idxs, "name", 0, "garren", 0, docId]
[idxs, "age", 0, 20, 0, docId]
[idxs, "location.country", 0, "RSA", 0, docId]
[idxs, "location.home.address", 0, "25 Sunny Road", 0, docId]
[idxs, "interests", 0, "rugby", 0, docId]
[idxs, "interests", 0, "music", 1, docId]
[idxs, "interests", 0, "climbing", 2, docId]
```

Document Metadata

```
[idxs, "_tigris_created_at",0,"2023-01-16T12:55:17.304154Z", 0, d
[idxs, "_tigris_updated_at",0,"2023-01-16T12:55:17.304154Z", 0, d
```

Queries

```
userCollection.findMany({
    filter:{
        "name": "garren",
        "age": {"$gte": 20},
        "location.country": {"$eq": "RSA"},
        "interests": {$in: ["climbing", "music"]}
    }
})
```

Query Planner

Query Planner

- Start simple and build it out as we go

Query Planner

- Start simple and build it out as we go
- Prefers equals to range query

Query Planner

- Start simple and build it out as we go
- Prefers equals to range query
- Prefers bounded range query to full range query

Query Planner

- Start simple and build it out as we go
- Prefers equals to range query
- Prefers bounded range query to full range query
- Will fall back to a table scan if the field is not indexed

Should We Auto Index Everything?

Should We Auto Index Everything?

1. Experienced Devs are use to adding indexes
2. A performance decrease in writes without the developer knowing why
3. Will revisit this later with helpful options for beginners

2. Building An Index

2. Building An Index

1. Documents added after the index change are indexed in the same transaction

2. Building An Index

1. Documents added after the index change are indexed in the same transaction
2. Indexes are consistent with the primary data

2. Building An Index

1. Documents added after the index change are indexed in the same transaction
2. Indexes are consistent with the primary data
3. The new index is initially built in the background

Background Indexing

Background Indexing

1. Background building using a queue - based off the QuiCK paper

Background Indexing

1. Background building using a queue - based off the QuiCK paper
2. Index up to watermark

Background Indexing

1. Background building using a queue - based off the QuiCK paper
2. Index up to watermark
3. This is a work in progress 🚧

Queue Design

```
[queue_subspace, vesting_time, priority, item_id] = job
```

Queue Design

Vesting time = The wall clock time when the item is visible to background workers

Queue Design

```
[1678281733333, 0, id1]
[1678281734444, 0, id2] <-- "Current Time"
[1678281735555, 0, id3]
[1678281735555, 1, id4]
[1678281736666, 0, id5]
```

Queue Design

```
    [1678281734444, 0, id2] <-- "Current Time"
    [1678281735555, 0, id3]
    [1678281735555, 1, id4]
    [1678281736666, 0, id5]
    [1678281888888, 0, id1] <-- "Back of the queue"
```

Queue Design

```
function enqueue(item: QueueItem, delay: Time)
function peak(maxItems: number): QueueItem[]

function obtainLease(item: QueueItem, leaseTime: Time)
function renewLease(item: QueueItem, leaseTime: Time)
function complete(item: QueueItem)
```

Background Worker

Background Worker

1. On schema change, index job is added to queue

Background Worker

1. On schema change, index job is added to queue
2. Mark index as write only

Background Worker

1. On schema change, index job is added to queue
2. Mark index as write only
3. Worker Peak and gets index job

Background Worker

1. On schema change, index job is added to queue
2. Mark index as write only
3. Worker Peak and gets index job
4. Batch Fetch items from FDB up to current time

Background Worker

1. On schema change, index job is added to queue
2. Mark index as write only
3. Worker Peak and gets index job
4. Batch Fetch items from FDB up to current time
5. Process items and renew lease

Background Worker

1. On schema change, index job is added to queue
2. Mark index as write only
3. Worker Peak and gets index job
4. Batch Fetch items from FDB up to current time
5. Process items and renew lease
6. Process next set of items and renew lease

Background Worker

1. On schema change, index job is added to queue
2. Mark index as write only
3. Worker Peak and gets index job
4. Batch Fetch items from FDB up to current time
5. Process items and renew lease
6. Process next set of items and renew lease
7. Complete job, mark index as ready

Handling Schema Upgrades

Handling Schema Upgrades

1. Remove field will delete the index

Handling Schema Upgrades

1. Remove field will delete the index
2. Add a field will index the field

Handling Schema Upgrades

1. Remove field will delete the index
2. Add a field will index the field
3. Changed field type

Handling Schema Upgrades

```
[idxs, ...key_path, version, value, dup_key, doc_id]
[idxs, "age", 0, "20", 0, docId]
[idxs, "age", 1, 20, 0, docId]
```

Index Statistics And Hot Spots

Index Statistics And Hot Spots

- Keep index metadata via atomic updates

Index Statistics And Hot Spots

- Keep index metadata via atomic updates
- Avoids write conflicts

Metadata Hot Spots

```
[idxs, info, count] = number of rows
[idxs, info, size] = Size of index
[idxs, path_count, ...key_path] = number of rows
```

Metadata Hot Spots

Metadata Hot Spots

- This lead to a much slower throughput

Metadata Hot Spots

- This lead to a much slower throughput
- Always be performance testing

Metadata Hot Spots

- This lead to a much slower throughput
- Always be performance testing
- Replace with GetEstimatedRangeSizeByte

Future Work

1. Unique indexes
2. Complex filtering
3. Smarter query planner

Find Us On Github

https://github.com/tigrisdata/tigris

Sign up for the Beta or join the discord community

https://www.tigrisdata.com/

Thank You And Any Questions?

@garrensmith

https://www.tigrisdata.com/