Mongo query execution

Overview

Mongo follows the well-known Volcano iterator model for query execution. It creates a parse tree followed by an execution tree, consisting of nodes which retrieve records from the storage layer and transform it before returning them to the user via the root of the tree.

The parsed query is represented by a tree of QuerySolutionNodes. This is transformed into a tree of PlanStages which is executed by the PlanExecutor.

Each PlanStage can have zero or more input streams but only one output stream. The root node calls work() to initiate work down the tree.

Data access to the storage layer is done at leaf nodes. The leaf stages are

1. CollectionScan (for data obtained directly from collection without matching index)
2. IndexScan (for data obtained from index)
3. QueuedDataStage (a pseudo-stage which returns data pushed into a queue)

Working Set

All Stages use a common WorkingSet which consists a set of records. These records transition through three stages (LOC_AND_IDX, LOC_AND_OBJ, OWNED_OBJ)

1. An Index scan returns a record id in LOC_AND_IDX satte
2. A Collection Scan returns a working set id in LOC_AND_OBJ state
3. An OWNED_OBJ represents a record that has been invalidated or doesnt correspond to on-disk obj

PlanStage::invalidate() & doInvalidate() is called to notify when a Record is going to be deleted so that all stages in the tree can invalidate any associated state.

Description of individual nodes in the query/plan tree

The various nodes in the query parse tree andthe corresponding nodes in plan execution tree are as follows:

In the syntax **"A -> B**" below, "A" indicates QuerySolutionNode(parse tree), "B" indicates PlanStage(plan execution node)

The number of PlanStage-type nodes is more than QuerySolutionNodes because execution adds more operations to the actual query.

FetchNode -> FetchStage : It contains the filter predicate MatchExpression. In case a suitable index is not available for this query, this node is added above a collection scan node to do filtering (in planner_access.cpp)

SkipNode -> SkipStage : A skip clause in the query creates the to skips first n records.

LimitNode -> LimitStage : Created by limit clause in query to limit result set to n records.

AND nodes : An AND clause in the query creates the AND node to do index intersection of two or more index scans. (MatchExpression::AND)

1. if sort by disk location is desired, then create **AndSortedNode -> AndSortedStage**
2. else sort by hash which creates AndHashNode -> AndHashStage

OR nodes : An OR clause in the query creates the OR node over two or more index scan (MatchExpression::OR)

1. if the sort order of child nodes is different, then create MergeSortNode -> MergeSortStage
2. if sort order is same -> OrNode -> OrStage

SubplanStage - can OR queries be run as sub-queries

I**ntermediate sort** : if index scan cannot fulfill requested sort order of the query

SortNode -> SortStage (has SortKeyGeneratorStage as child to supply its sort keys)

SortKeyGeneratorNode -> SortKeyGeneratorStage

ProjectionNode -> ProjectionStage : If a projection is desired

DistinctNode -> DistinctScan : if only distinct values are to be returned

Count nodes : If only count is desired

CountNode -> CountScan

CountStage - getExecutorCount

Group by condition : (db.coll.group())

GroupStage - created by GroupCommand

For geo-spatial queries, special nodes are created

GeoNear2DNode -> GeoNear2DStage

GeoNear2DSphereNode -> GeoNear2DSphereStage

NearStage - geo index

Return deleted or updated docs : To merge docs which were deleted or updated during the query, mutations node are used

KeepMutationsNode -> KeepMutationsStage

Text search : For text search, the following nodes are used

TextNode -> TextStage

TextMatchStage : returns documents which match full-text search, created by TextStage::buildtextTree

TextOrStage - unused

Delete : To delete a doc fetched by lower stage node,

DeleteStage - created by getExecutorDelete

Update: For updates/upsert to a doc

UpdateStage - created by getExecutorUpdate

EOFStage : For collections which dont exist and for end of plan execution, use EOFStage

ID index : To use default "_id" index, IDHackStage node is used

Iterator (not clear)

IndexIteratorStage - wrapper around cursor SortedDataInterface

MultiIteratorStage - special stage used in ParallelCollectionScanCmd and repairCursor

createRandomCursorExecutor in pipeline calls creates these nodes

Plan cache access: Mongo caches plans and seems to have special nodes in the execution tree for them

CachedPlanStage - to extract a plan from the PlanCache

MultiPlanStage - when there are many possible plans, help choose one; CachedPlanStage::replan

OplogStart – hack used in replication to walk backwards in a collection to find a document which matches query

PipelineProxyStage - PipelineCommand

ShardingFilterNode -> ShardFilterStage (to drop docs which are not part of shard)

Expression parser

The expressions in a Mongo query are represented by the MatchExpression class and its derived classes. Complex expressions are represented by a tree of MatchExpressions.

fromjson converts : query string -> BSONObj

MatchExpressionParser converts : BSONObj -> MatchExpression

The derived classes of MatchExpression are as follows:

(*indentation signifies derived class)*

(suffix MatchExpression omitted in some names)

```
MatchExpression
 ArrayMatchingMatchExpression
   ElemMatchObject
   ElemMatchValue
   SizeMatch
```

- AtomicMatchExpression
- FalseMatchExpression
- LeafMatchExpression
  - GeoMatch
  - GeoNearMatch
  - Regex, ModMatch, Exists, In
  - BitTest
    - BitAllSet, BitAllClear, BitAnySet, BitAnyClear
  - Comparison
    - LT, LTE, GT, GTE, Equality
  - TextMatchExpressionBase
    - TextNoOp
- ListOfMatchExpression
  - And, Or
- NotMatchExpression
- TypeMatchExpression
- WhereMatchExpressionBase
  - WhereMatchExpression
  - WhereNoOpMatchExpression