\*Doesn’t include fault tolerance piece yet or reduce

**Service provided by Node via gRPC**

message TaskRequest {

string obj\_id = 1; //object id of the to-be created object

int32 task\_id = 2;

repeated string args = 3; //List of arguments as objId’s associated with the task

map<string, bytes> obj\_id\_to\_obj = 4; //Pre-specified params (“generic” type)

}

message TaskResponse { }

message TaskAnsRequest {

string obj\_id = 1;

}

message TaskAnsResponse {

bytes res = 1;

}

message OdsInfo {

int64 size = 1; //The full expected size of the object upon completion

map<string, bool> location\_infos = 2; //node and completion status pairing (1 if complete, 0 if partial)

}

message OdsGetRequest {

string key = 1; //object id

}

message OdsGetResponse {

OdsInfo value = 1;

bool was\_found = 2;

}

message OdsSetRequest {

string key = 1;

OdsInfo value = 2; //LocationInfos should have 1 element in this case

int64 ttl\_ms = 3;

}

message OdsSetResponse {}

message OdsDeleteRequest {

string key = 1;

string node\_name = 2;

}

message OdsDeleteResponse {}

message BroadcastObjRequest {

string object\_id = 1;

int32 start = 2; //where to start copying from (to allow for partial copy transmission)

}

Message BroadcastObjResponse {

bytes object = 1;

}

message DeleteObjRequest {

string object\_id = 1;

}

Message DeleteObjResponse {

}

service TaskDistr {

//RPC calls between nodes for Ods

rpc OdsGet(OdsGetRequest) returns (OdsGetResponse);

rpc OdsSet(OdsSetRequest) returns (OdsSetResponse);

rpc OdsDelete(OdsDeleteRequest) returns (OdsDeleteResponse);

//RPC calls between nodes for Ods

rpc BroadcastObj(BroadcastObjRequest) returns (BroadcastObjResponse)

rpc DeleteObj(DeleteObjRequest) returns (DeleteObjResponse)

//RPC call between TaskScheduler and nodes

Rpc ScheduleTask(TaskRequest) returns (TaskResponse);

Rpc GetTaskAns(TaskAnsRequest) returns (TaskAnsResponse);

}

**Task scheduler**

**Data**

type TaskScheduler {

nodeBusy []bool

objIds []string //list of used objIds

clientPool ClientPool

shutdown   chan struct{}

mu         sync.RWMutex

}

**Functions**

func MakeTaskScheduler(clientPool ClientPool, doneCh chan struct{}) \*TaskScheduler

Make + return TaskScheduler object

**TaskScheduler Methods**

ScheduleTask(taskId int32, args string[], objIdToObj map<string, []byte>) (string, err)

//Async: returns object iD (a promise)

Create a new unused objectId

go ScheduleTaskHelper

return objectId

ScheduleTaskHelper(taskId int32, args string[], objIdToObj map<string, []byte>, chan []byte)

Wait for free node, set node status to busy

Send grpc request TaskRequest and wait for response. ((If err, try again on diff node))

Set chosen node status to free

RetrieveObject(objId string) ([]byte, err)

//Synchronous: fetch object when it becomes available

Choose random node

Send grpc request GetTaskAns, return response

**Node**

**Data**

type OdsShard {

data  map[string]\*proto.OdsInfo //objId → objInfo; set to nil if shard not currently active

mu   sync.RWMutex

}

type Ods {

shardMap   \*ShardMap

ourShards  map[int]struct{}

clientPool ClientPool

shard      []KvShard

}

type LocalObj{

data []byte

mu    sync.RWMutex

}

type LocalObjStore{

mp   map<string, localObj>

mu    sync.RWMutex

}

type Node {

//GENERAL

nodeName string

shutdown   chan struct{}

//OBJECT STORE

localObjStore LocalObjStore

//ODS SHARD MAP

ods Ods

}

**Node Methods (general)**

Shutdown()

MakeNode(nodeName string, shardMap \*ShardMap, clientPool ClientPool) \*Node

**Node Methods (associated w/ worker)**

RunTask(ctx context.Context, request \*proto.TaskRequest) (\*proto.TaskResponse, error) //Called by TaskScheduler via RPC

Basically a switch case between all the possible tasks

SampleTask(x1Id string, x2Id string, retObjId string, objIdToObj map<string, []byte>)

//Sample task that just adds two numbers

Run task, using GetGlobalObject() to fetch objects when necessary

Use PutGlobalObject() to store result

RetrieveTaskAns(ctx context.Context, request \*proto.TaskAnsRequest) (\*proto.TaskAnsResponse, error)

//Called by TaskScheduler via RPC

Returns GetObject(the requested objectId, nil)

**Node Methods (associated w/ object management)**

GetGlobalObject(objId string, objIdToObj map<string, []byte>) ([]byte)

//Retrieves object from object Id: synchronous

If objectId in the pre-specified params provided by task manager, return it

If object is stored on this node’s localObjectStore, return it

While object not complete:

Find object location through OdsGetReq; choose the complete copy (if available) or the largest partial copy otherwise

Retrieve object (or missing section of it) w/ gRPC BroadcastObjRequest call

Save object in localObjectStore for future reference before returning it

PutGlobalObject(objId string, obj []byte)

//Creates object with object Id in local object store, updates ODS

Save obj with objId to localObjStore

Send a OdsSetReq

DeleteGlobalObject(objId string)

//Deletes object from all nodes

Find object locations through OdsGetReq

For each object location, send a grpc call DeleteObjRequest

BroadcastLocalObject(ctx context.Context, request \*proto.BroadcastObjRequest) (\*proto.BroadcastObjResponse, error)

Retrieve and return appropriate object in localObjStore

DeleteLocalObject(ctx context.Context, request \*proto.DeleteObjRequest) (\*proto.DeleteObjResponse, error)

//Deletes local copy of the object specified by the provided object Id

Remove specified object from LocalObjStore

ReduceObject

TBD

**Node Methods (associated w/ ODS)**

//ODS CALLS SENDING REQUESTS (KV Client)

OdsGetReq(ctx context.Context, key string) (string, bool, error)

OdsSetReq(ctx context.Context, key string, value \*proto.OdsInfo) error

OdsDeleteReq(ctx context.Context, key string) error

//ODS CALLS GENERATING RESPONSES (KV Server)

OdsGetRes(ctx context.Context, request \*proto.OdsGetRequest) (\*proto.OdsGetResponse, error)

OdsSetRes(ctx context.Context, request \*proto.OdsSetRequest) (\*proto.OdsSetResponse, error)

OdsGetRes(ctx context.Context, request \*proto.OdsDeleteRequest) (\*proto.OdsDeleteResponse, error)