findOrCreate operation in Mongo

TERRA frequently encounters following situations:

- two different extractors process two different files that belong in the same destination dataset, at the same instant
- two different extractors process two different datasets that belong in the same destination collection, at the same instant

...both extractors query Clowder for the existence of the dataset/collection by name at the same time, and Clowder responds to both saying "does not exist". Both extractors then create the dataset/collection and upload their data to it, and because Clowder does not enforce unique dataset/collection names we end up with duplicate datasets/collections, each having part of the necessary data.

What we see in TERRA are two identical collections, one with e.g. 8000 datasets and the other with just 1 - they were both created at once, but for all subsequent extractors the 8000 collection is the one returned and the other one is basically orphaned. I have a script we run periodically to identify these duplicates and merge them and delete the leftover empty one, but this is not ideal.

Rob Kooper found this topic: https://stackoverflow.com/questions/16358857/mongodb-atomic-findorcreate-findone-insert-if-nonexistent-but-do-not-update/16362833#16362833

> Beginning with MongoDB 2.4, it's no longer necessary to rely on a unique index (or any other workaround) for atomic `findOrCreate` like operations.
>
> This is thanks to the `$setOnInsert` operator new to 2.4, which allows you to specify updates which should only happen when inserting documents.
>
> This, combined with the `upsert` option, means you can use `findAndModify` to achieve an atomic `findOrCreate`-like operation.
>
> ```
> db.collection.findAndModify({
> query: { _id: "some potentially existing id" },
> update: { $setOnInsert: { foo: "bar" } }, new: true, // return new doc if one is upserted
> upsert: true // insert the document if it does not exist })
> ```
>
> As `$setOnInsert` only affects documents being inserted, if an existing document is found, no modification will occur. If no document exists, it will upsert one with the specified _id, then perform the insert only set. In both cases, the document is returned.

https://docs.mongodb.com/manual/reference/method/db.collection.findAndModify/

https://docs.mongodb.com/manual/reference/operator/update/setOnInsert/#op._S_setOnInsert

In our case I think pseudocode would be:

- query for dataset/collection with name exactly matching NEW_NAME
- in update clause, $setOnInsert will be the creation parameters (name, agent, parent/child relationships, etc) and these are only applied if the entity is not already found - a new object will be created with these parameters
- upsert: true will create the new object if it isn't found
- new: true will return the new ID if it is created
- otherwise, existing object is returned in query as normal

I think the race condition still exists here but because there's no call/response across 2 calls to Clowder API (before it was 1) does it exist? 2) no - create) the window for collisions is much smaller, like the same microsecond.

I also note this comment from SO thread:

> One needs to be careful here, though. This only works if the selector of the findAndModify/findOneAndUpdate/updateOne uniquely identifies one document by _id. Otherwise the upsert is split up on the server into a query and an update/insert. The update will still be atomic. But the query and the update together will not be executed atomically.