Bases de Données Avancées, Rapport Phase 3 : Replica Set

I. Introduction

Ce rapport correspond au compte-rendu de la troisième phase du cours de bases de données avancées. Nous verrons au cours de celui-ci comment créer un replica set pour garantir la persistance et la disponibilité de nos données en cas de panne ou autre imprévu.

II. Création des replica sets

A. Architecture

On utilisera une architecture à 3 bases de données : 1 principale (primary) et deux secondaires (secondary 1 et 2). En fonctionnement normal, les deux nœuds secondaires copieront passivement les opérations effectuées sur le principal (s1 copiera p et s2 copiera s1). En cas de panne du principal, le nœud secondaire 1 sera élu pour les opérations de lecture / écriture. En cas de panne du principal et du secondaire 1, le secondaire 2 sera élu, pour lecture uniquement. En effet, on choisit un quorum de 2/3 : au moins deux nœuds doivent être opérationnels pour l'écriture, afin de garantir la cohérence des données.

Notre architecture est donc :

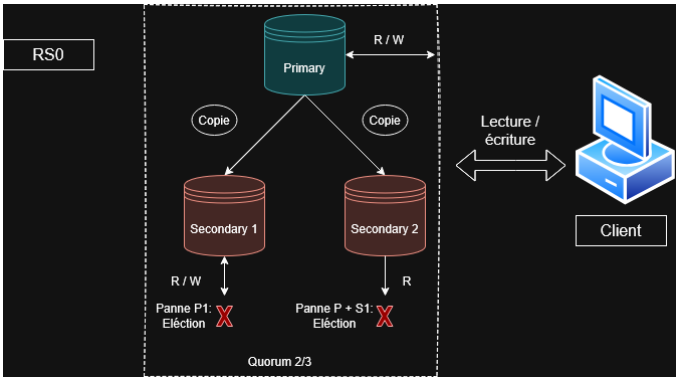

Figure 1. Architecture du replica set

B. Configuration

On lance 3 instances mongoDB avec mongod : une pour chaque nœud. On associe à chaque nœud un port et un répertoire pour y écrire les données différent :

Noeud	répertoire (dans data/mongo)	port
Primary	db-1	27017
Secondary 1	db-2	27018
Secondary 2	db-3	27019

Table I. Paramètres des nœuds

On lance les instances avec :

```
mongod --replSet rs0 --port 2701X --dbpath
↪  ./data/mongo/db-X --bind_ip localhost
```

dans des terminaux différents.

On se connecte au nœud principal avec :

```
mongosh --host localhost --port 27017
↪  --directConnection
```

Puis dans mongosh, on exécute :

```
rs.initiate({
    _id: "rs0",
    members: [
        { _id: 0, host: "localhost:27017" },
        { _id: 1, host: "localhost:27018" },
        { _id: 2, host: "localhost:27019" }
    ]
})
```

pour initialiser le replica set.

III. Test du replica set

A. état initial

On vérifie le statut du replica set avec `rs.status()` : Le log résultant est très long et contient énormément d'informations. Pour vérifier l'état de notre replica set on vérifie uniquement certains champs.

- "myState" indiquant si le replica set est actif.
- "majorityCount" donnant le quorum pour notre set, ici 2.
- "votingMembersCount" indiquant le nombre de votants, ici 3.

Dans l'array "members", on vérifie que l'on a bien :
Pour le premier membre :

```
    _id: 0,
    name: 'localhost:27017',
    health: 1,
    state: 1,
    stateStr: 'PRIMARY'
```

Indiquant que c'est bien le nœud principal qui a l'id 0 et qui est au port 27017. De plus le champs "health" confirme qu'il est actif et opérationnel.

On vérifie également que "syncSourceId" = -1 et "self" = true : c'est à dire, qu'il ne copie ses données d'aucun autre noeud et que c'est bien la base de données à laquelle on est connectés.

Pour le deuxième et troisième membre :

```
_id: {1 / 2},
name: 'localhost:{27018 / 27019}',
health: 1,
state: 2,
stateStr: 'SECONDARY',
```

Ainsi que : "syncSourceHost" = 'localhost :27017' et "syncSourceId" = 0, pour s'assurer qu'ils dupliquent bien les données du principal.

B. *écriture*

On vérifie à l'aide d'un script que les bases de données secondaires copient bien les données à l'insertion. Pour cela on crée une nouvelle connexion "test" dans la principale, et on y insère un élément. On vérifie ensuite si l'élément a bien été inséré dans les deux autres. Un test rapide nous montre que oui : la duplication fonctionne comme elle devrait.

C. *panne Primary*

On force l'arrêt de notre instance principale de mongod avec ctrl c pour vérifier si l'élection d'un principal parmi les secondaires a bien lieu.

Pour cela, on se connecte avec mongosh au nœuds secondaires, et on lance la commande rs.isMaster(). On obtient le résultat :

```
ismaster: true,
secondary: false,
primary: 'localhost:27018',
me: 'localhost:27018',
```

Indiquant que l'élection a bien eu lieu et que le nouveau primary est à présent db-2.

D. *nouveau Primary*

On cherche à présent à mesurer le temps que prends cette élection. Il est très important que cela est lieu le plus rapidement possible, pour que la base de données soit disponible le plus tôt possible après la panne d'un des nœuds.

Pour cela on crée un script qui mesure le temps d'élection. On trouve un temps d'élection de 0.0002 secondes environ, soit un temps suffisamment court pour que ça n'ait pas un impact fortement visible sur la disponibilité de la base de données.

E. *lecture*

On vérifie que comme prévu, les données sont toujours lisibles même avec un nœud en moins. On voit qu'en effet, nous avons bien accès aux données du nouveau primary.

F. *reconnexion*

On relance le nœud que l'on avait arrêté. On observe que le nœud qui avait nouvellement été élu primary (secondary 1) reste le primary plutôt que de le rendre à db-1.

G. *double panne*

Cette fois ci, on fait tomber le primary et un noeud secondaire en même temps. On se rend compte que si les données restent lisibles sur le dernier noeud, l'insertion n'est en revanche plus possible. Cela est dû au quorum de 2/3 de notre replica set.

IV. Conclusion

Notre replica set est à présent opérationnel et prêt à l'emploi. Nous avons pu nous assurer qu'il fonctionnait correctement et voir comment fonctionner l'élection du noeud principal en cas de panne. Il nous permettra de garantir la disponibilité des données de notre future application ainsi que leur intégrité même en cas d'imprévu.