ESILV LÉONARD DE VINCI | ÉCOLE D'INGÉNIEURS PARIS-LA DÉFENSE

# Scalability

# Développement d'application Cloud

Nicolas Picard
Henri Duhamel
Johan Van Der Slooten
Maxence Clusaz

Novembre 2019

# Table des Matières

# 1 Importation des données

Tout d'abord, nous avons dû mener une réflexion quant au langage de programmation à utiliser, de manière à exporter et formatter le plus rapidement possible la base de données existante en documents. Les options les plus adaptées étaient Python, C# et Golang, ce dernier étant très réputé pour ses capacités en multithreading.

Nous avons finalement choisi d'utiliser Python pour sa simplicité et son système de modules. En effet, les dictionnaires Python et les modules mysql-connector et pymongo nous ont été d'une grande aide. Ceux-ci étant assez flexibles, nous avons généré différentes méthodes permettant de formater les résultats obtenus depuis la base de données MySQL en dictionnaires, pour les insérer à l'aide de pymongo. Nous aurions dû créer un certain nombre de structures en Golang (tout comme en C# excepté que ce serait des objets et non des structures) ce qui n'était pas le plus pratique.

Nous avons utilisé le module multiprocessing de Python afin de mettre en parallèle le plus possible d'opérations et alors d'exporter plus rapidement la base entière. Selon nos calculs, nous avons divisé notre temps d'exportation par 125 en utilisant le multiprocessing.

Nous avons rencontré différents problèmes lors des premiers essais. En effet, nous avons régulièrement obtenu des timeouts dû au grand nombre de requêtes sur la base de données du serveur de FIT. Nous avons donc installé MySQL en local sur l'une des VM et copié la base de données de FIT à l'aide d'un dump SQL. En réalisant les requêtes en local, cela a permis d'éviter ces timeouts et ainsi la perte de données.

Nous avons également rencontré des problèmes lors du scaling en augmentant le nombre de shards. En effet, selon le nombre de shards disponibles, certains pouvaient avoir plus de chunks que d'autres, les données non équlibrées et ainsi certaines requêtes plus lentes, en particulier lorsque nous avons utilisé un nombre de shards impair.

Hormis ces petits problèmes, l'importation s'est bien passée et notre clé de sharding a bien été choisie puisqu'elle permet de bien répartir les employés selon leur département et accélérer les requêtes "communes" qui ciblent bien souvent un département spécifique. Les requêtes "analyste" étant assez lourdes (MapReduce), elles ont régulièrement pour clés les départements ce qui permet également d'accélérer celles-ci. Nous avons utilisé comme indexes _id_ ainsi que _current_dept.dept_no_, qui correspond à l'indexation de notre clé de sharding. Dans notre version finale, on dispose de 6 shards, ce qui répartit ainsi les données en 7 chunks, 1 par shard dont un qui en contiendra 2.

Après avoir revu notre cas d'usage, nous estimions qu'il était nécéssaire de rajouter certaines requêtes qui pouvaient être pertinentes auprès d'un analyste et d'un administrateur de bases de données. Nous avons donc ajouté les requêtes suivantes pour un analyste:

- Obtenir l'âge moyen des employés pour chaque département
- Obtenir la répartition du genre des employés par département
- Obtenir la répartition du genre des managers

Nous avons également revu les requêtes administrateur pour y ajouter (ces requêtes peuvent être combinées en une seule à l'aide de la fonction **stats()**):

- Obtenir le nombre de documents par shard
- Obtenir le nombre de shards
- Obtenir le nombre de chunks par shard
- Obtenir le poids moyen d'un document

# 2 Requêtes MongoDB

## 2.1 Requêtes utilisateur

```
opMatch = { $match: {"emp_no": 10001} } // Premier match sur l'employé
opUnwind = { $unwind: "$salaries" }     // Unwind des salaires afin d'obtenir celui de l'année actuelle
opProject = { $project: {emp_no: 1,
                         toDateYear: { $year: "$salaries.to_date" },
                         amount: "$salaries.salary"} }
// Projection pour obtenir le numéro d'employé, l'année to_date du salaire, et le salaire lui-même
opMatch2 = { $match: {"toDateYear" : { $eq: 9999 }} }
// Second match pour obtenir le salaire de l'année actuelle (année 9999 => en cours)

db.employees.aggregate([
    opMatch, opUnwind, opProject, opMatch2
])
```

Figure 1: Requête n°1 - Obtenir sa fiche de paie de l'année actuelle

```
db.employees.find({ "current_dept.dept_no": "d004" },
                  { "emp_no": 1, "first_name": 1, "last_name": 1 })
// Renvoie le num. employé, le nom/prénom des employés du département
```

Figure 2: Requête n°2 - Obtenir les employés travaillant dans son département

```
opMatch = { $match: {"current_dept.dept_no": "d007"} } // Premier match sur le département
opUnwind = { $unwind: "$titles" } // Unwind des titres de chacun pour trouver son titre actuel par la suite
opProject = { $project: {emp_no: 1,
                         currentDept: "$current_dept",
                         firstName: "$first_name",
                         lastName: "$last_name",
                         toDateYear: { $year: "$titles.to_date" },
                         isManager: "$titles.isManager"} }
// Projection des informations principales désirées ainsi que de l'année to_date
opMatch2 = { $match: {"toDateYear" : { $eq: 9999 }, "isManager": true} }
// Second match pour obtenir le titre actuel (année 9999 => titre actuel)

db.employees.aggregate([
    opMatch, opUnwind, opProject, opMatch2
])
```

Figure 3: Requête n°3 - Obtenir les managers d'un département

```
opMatch = { $match: { "current_dept.dept_no": "d004",
                      "current_dept.from_date" : {"$gte": new Date("2002-01-13T00:00:00.000+0000")} } };
// Premier match sur le département et la date d'arrivée dans le département
opProject = { $project: { "first_name": 1, "last_name": 1 } }
// Projection sur le nom/prénom
opSort = { $sort: { "current_dept.from_date": -1 } }
// Tri par date décroissant

db.employees.aggregate([
    opMatch, opProject, opSort
])
```

Figure 4: Requête n°4 - Obtenir les derniers employés arrivés dans un département

## 2.2 Requêtes analyste

```
mapFunction = function () {
    sum = 0; nb = 0;
    for(i=0; i < this.salaries.length;i++) {
        sum += this.salaries[i].salary
        nb++
    }
    // Emit de la somme des salaires et du nb de salaires
    emit(this.current_dept.dept_no, {"sum": sum, "nb": nb})
}
reduceFunction = function (key,values) {
    sum = 0; nb = 0;
    for (i=0; i < values.length; i++) {
        sum += values[i].sum
        nb += values[i].nb
    }
    // Calcule la moyenne des salaires par dept
    return {"avg": sum/nb, "sum": sum, "nb": nb};
};
queryParam = {"query":{}, "out":{"inline":true}};
db.employees.mapReduce(mapFunction, reduceFunction, queryParam);
```

Figure 5: Requête n°5 - Obtenir le salaire moyen par département

```
mapFunction = function () {
    emp_no = this.emp_no
    for(i=0; i < this.titles.length;i++) {
        if(this.titles[i].to_date.getYear() + 1900 === 9999) {
            // emit du numéro d'employé avec pour clé son titre actuel
            emit(this.titles[i].title, {"emp_no": [emp_no] })
        }
    }
}
reduceFunction = function (key,values) {
    emps = []
    for(i=0;i<values.length; i++) {
        for(j=0;j<values[i].emp_no.length;j++) {
            emps.push(values[i].emp_no[j])
        }
    }
    // Retourne les listes d'employés ayant le même titre
    return {"emp_no": emps};
};
queryParam = {"query":{}, "out":{"inline":true}};
db.employees.mapReduce(mapFunction, reduceFunction, queryParam);
```

Figure 6: Requête n°6 - Obtenir les employés ayant le même titre

```
mapFunction = function () {
    birthyear = this.birth_date.getFullYear() // année de la date de naissance
    todayyear = new Date().getFullYear() // année actuelle
    age = todayyear - birthyear
    // emit de l'age pour le département et du nb pour faire la moyenne
    emit(this.current_dept.dept_no, {"age": age, "nb": 1})
}
reduceFunction = function (key,values) {
    sum = 0; nb = 0;
    for (i=0; i < values.length; i++) {
        sum += values[i].age
        nb += values[i].nb
    }
    // simple calcul de moyenne de l'âge
    return {"avg": sum/nb, "age": sum, "nb": nb};
};
queryParam = {"query":{}, "out":{"inline":true}};
db.employees.mapReduce(mapFunction, reduceFunction, queryParam);
```

Figure 7: Requête n°7 - Obtenir l'age moyen des employés pour chaque département

```
mapFunction = function () {
    // emit du genre avec le nb et le dept pour clé
    if(this.gender === "M") {
        emit(this.current_dept.dept_no, {"M": 1, "F": 0, "nb": 1})
    } else {
        emit(this.current_dept.dept_no, {"M": 0, "F": 1, "nb": 1})
    }
}
reduceFunction = function (key,values) {
    sumM = 0; sumF = 0; nb = 0;
    for (i=0; i < values.length; i++) {
        sumM += values[i].M
        sumF += values[i].F
        nb += values[i].nb
    }
    // retourne le nombre d'hommes et de femmes par département
    return {"M": sumM, "F": sumF, "nb": nb};
};
queryParam = {"query":{}, "out":{"inline":true}};
db.employees.mapReduce(mapFunction, reduceFunction, queryParam);
```

Figure 8: Requête n°8 - Obtenir la répartition du genre des employés par département

```
mapFunction = function () {
    emp_no = this.emp_no
    for(i=0; i < this.titles.length;i++) {
        if(this.titles[i].to_date.getYear() + 1900 === 9999 && this.titles[i].isManager) {
            // emit du nb et du genre en clé pour classer par genre
            emit(this.gender, {"nb": 1})
        }
    }
}
reduceFunction = function (key,values) {
    // simple retour de longueur de values
    return {"nb": values.length};
};
queryParam = {"query":{}, "out":{"inline":true}};
db.employees.mapReduce(mapFunction, reduceFunction, queryParam);
```

Figure 9: Requête n°9 - Obtenir la répartition du genre des managers

### 2.3 Requêtes administrateur

```
db.employees.stats() // get collection stats
```

Figure 10: Requête n°10 - Obtenir des statistiques sur la base de données

Cette requête est on ne peut plus simple, mais c'est uniquement car elle renvoie beaucoup d'informations à propos de la base de données et de la collection. En effet, à partir de cette requête, on peut extraire:

- Le nombre de documents par shard
- Le nombre de shards
- Le nombre de chunks par shard
- Le poids moyen d'un document
- La taille de la collection

Ces informations sont à nos yeux les plus importantes à extraire pour un administrateur base de données. Il suffit donc de réaliser un léger traitement applicatif récupérant l'essentiel de cette requête pour les renvoyer par la suite à l'administrateur.

# 3 Performance

Comme nous disposions de 8 VMs, nous avons décidé d'effectuer nos tests en utilisant 1 routeur mongos, 1 config server, et de 1 à 8 shards, sachant qu'un réplicatSet était 1 shard. Vous trouverez ci-dessous les résultats des tests de performance qui ont été effectués sur chacune des requêtes de notre cas d'usage. Nous avons pour chaque cas, réalisé 12 mesures, enlevé la plus grande et la plus petite pour obtenir 10 mesures fiables.

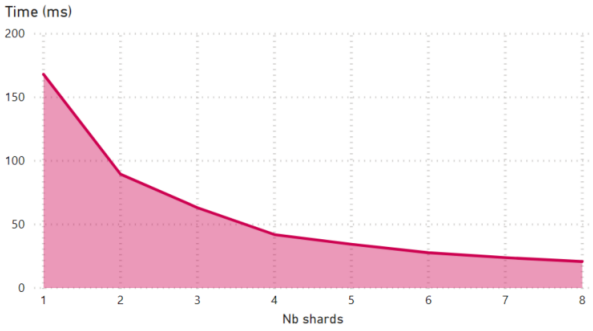


Figure 11: Requête n°1

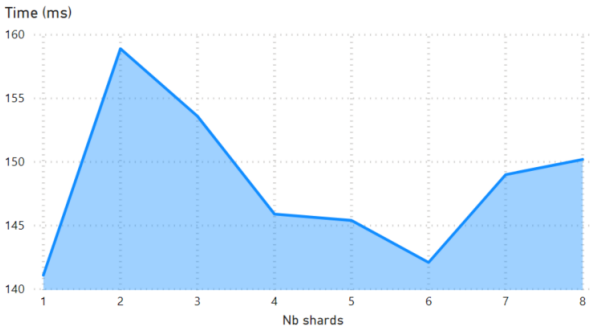


Figure 12: Requête n°2

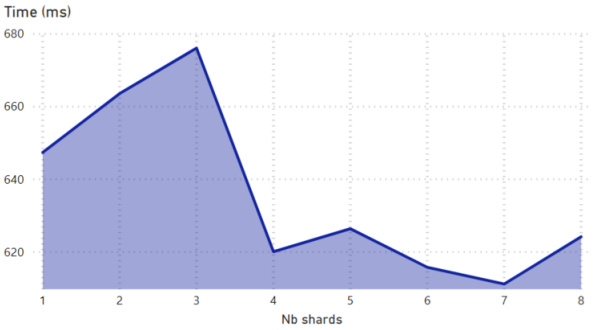


Figure 13: Requête n°3

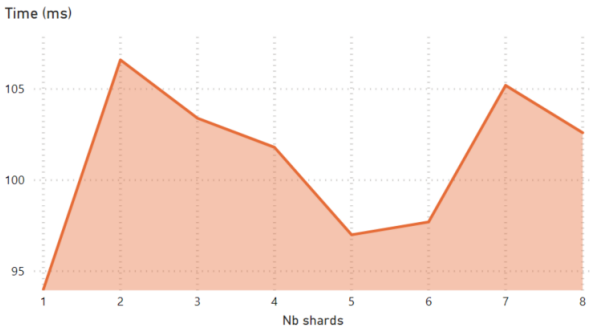


Figure 14: Requête n°4

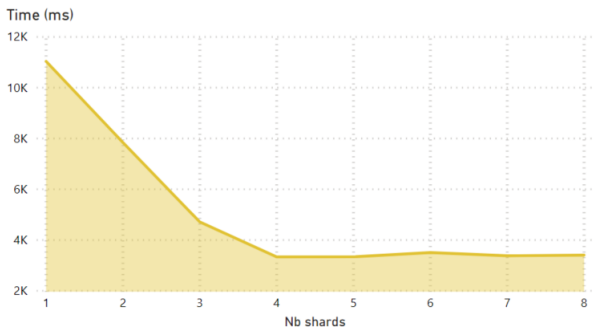


Figure 15: Requête n°5

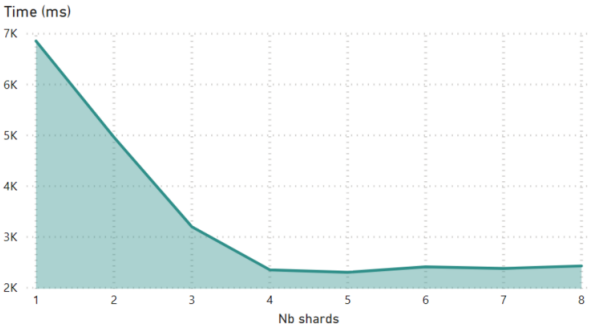


Figure 16: Requête n°6

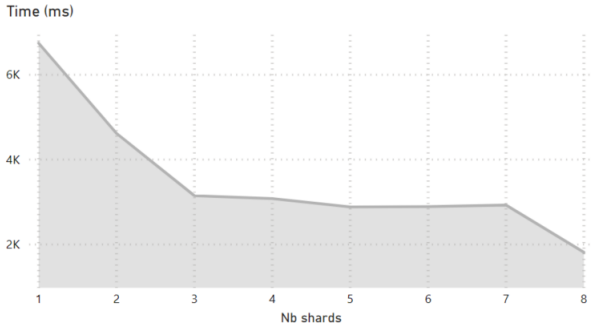


Figure 17: Requête n°7

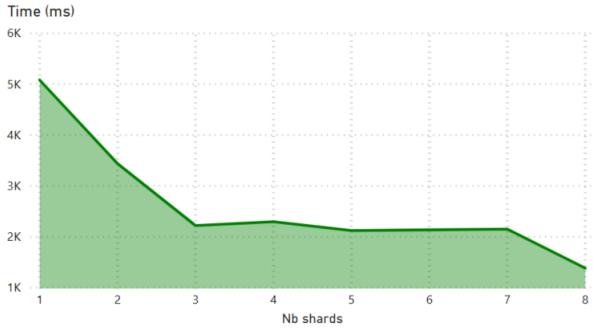


Figure 18: Requête n°8

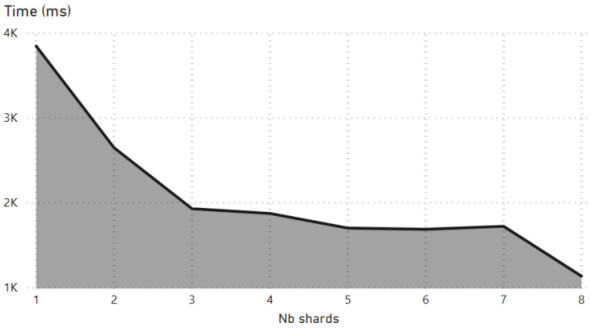


Figure 19: Requête n°9

# 4 Conclusion

Après avoir réalisé de nombreux tests de performance sur chacune des requêtes et ce avec un nombre croissant de shards, nous pouvons conclure que nous choisirons d'utiliser 6 shards. En effet, vous pouvez voir qu'en utilisant la méthode **db.collection.getShardDistribution()**, on voit qu'un shard est vide, seuls 7 des shards possèdent réellement des données. Dans le cas où on dispose de 8 shards, les données sont décomposées en 9 chunks, tandis qu'avec 6 shards, on dispose de 7 chunks.

```
Totals
 data : 251.95MiB docs : 300024 chunks : 9
 Shard RS1 contains 6.26% data, 5.85% docs in cluster, avg obj size on shard : 941B
 Shard RS2 contains 4.47% data, 4.29% docs in cluster, avg obj size on shard : 915B
 Shard RS4 contains 21.83% data, 24.81% docs in cluster, avg obj size on shard : 774B
 Shard RS3 contains 21.11% data, 20.46% docs in cluster, avg obj size on shard : 908B
 Shard RS5 contains 0% data, 0% docs in cluster, avg obj size on shard : 0B
 Shard RS6 contains 18.33% data, 17.71% docs in cluster, avg obj size on shard : 911B
 Shard RS7 contains 9.46% data, 9.09% docs in cluster, avg obj size on shard : 916B
 Shard RS8 contains 18.52% data, 17.76% docs in cluster, avg obj size on shard : 918B
```

Figure 20: Répartition des données avec 8 shards

```
Totals
 data : 251.95MiB docs : 300024 chunks : 7
 Shard RS6 contains 18.52% data, 17.76% docs in cluster, avg obj size on shard : 918B
 Shard RS1 contains 30.63% data, 33.35% docs in cluster, avg obj size on shard : 808B
 Shard RS2 contains 6.26% data, 5.85% docs in cluster, avg obj size on shard : 941B
 Shard RS3 contains 18.33% data, 17.71% docs in cluster, avg obj size on shard : 911B
 Shard RS4 contains 26.24% data, 25.3% docs in cluster, avg obj size on shard : 913B
 Shard RS5 contains 0% data, 0% docs in cluster, avg obj size on shard : 0B
```

Figure 21: Répartition des données avec 6 shards

Compte tenu des problèmes que nous avons rencontré lors de la répartition des chunks, nous utiliserons six shards, un config server et un routeur mongos. En effet, d'après les graphes de performance on peut observer une bonne performance pour la plupart des requêtes lorsqu'on dispose de 6 shards (à noter que dans notre cas, 1 réplica $\Leftrightarrow$ 1 shard).

De plus, en augmentant le nombre de shards, nous répartissons beaucoup plus les données de chaque département et donc provoquons du trafic réseau. Dans un cadre de production, il pourrait être plus efficace d'utiliser moins de shards, mais plus de routeurs, config servers afin d'augmenter notre tolérance aux pannes, ainsi qu'avoir plus de réplicas afin d'assurer un service continu si jamais un serveur primary au sein d'un réplicaSet venait à tomber. Lorsque l'on développera l'application web, nous pourrions utiliser un système de cache afin de réduire les requêtes fréquentes (les requêtes utilisateur).

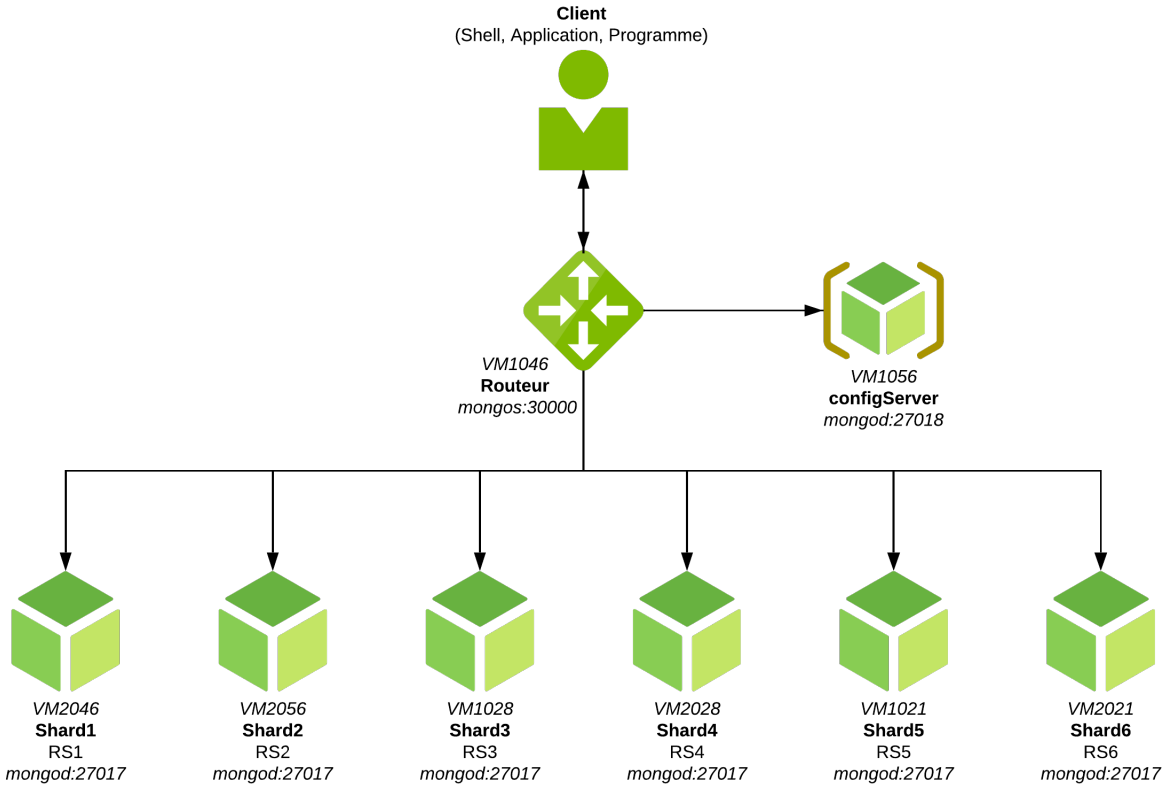


Figure 22: Diagramme de l'architecture MongoDB