Technologies Logicielles pour le Cloud computing

Google Data Store

Hugo BANNIER - Riwal LE BUHAN - Gwénolé LE HENAFF

ILa 2018/2019

1. Introduction

Au cours de notre TP de TLC, nous avons pu mettre en place un serveur utilisant le cloud de Google avec notamment la gestion de la base de donnée Google Datastore et l'AppEngine pour le back-end Java. Pour se faire nous avons donc mis en place une API REST de Fitness.

Github : https://github.com/ISTIC-M2-ILa-GM/TLCFitness
Url de l'application : https://tlcfitness.appspot.com
Benchmark : https://drive.google.com/file/d/1UXfwmMvNLIMfoz3ZWrnXa3Z-dCL1CEF-/view

Nous tenons à remercier notre professeur, Quentin DUFOUR pour toute l'aide et toute l'attention qu'il a pu nous fournir.

2. Benchmark

Nous avons utilisé le framework Gatling pour nous permettre d'en ressortir des graphiques.

Code de test

Gatling utilise des scripts écrits en Scala qui restent plutôt simple et intuitif à lire.

Voici donc un exemple du benchmark que nous avons effectué :

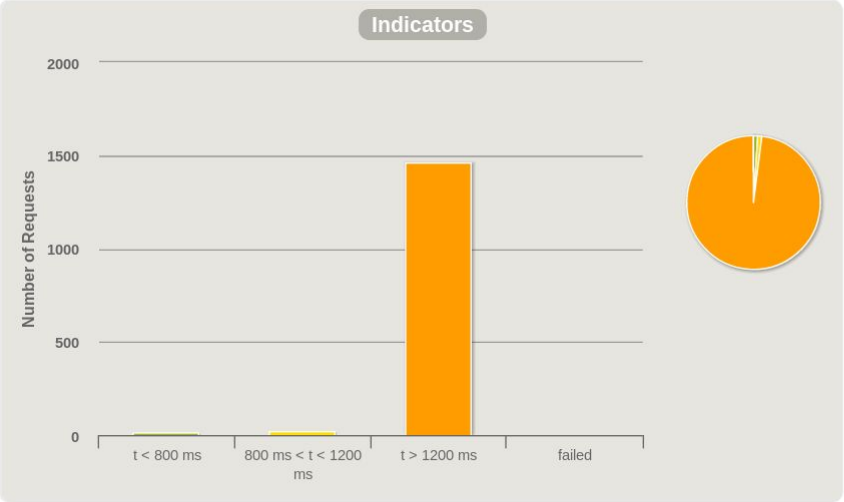
```
class FitnessSimulationTest extends Simulation {

 val httpConf = http.baseUrl("https://tlcfitness.appspot.com")

 val scn = scenario("FitnessSimulation")
   .exec(http("add")
     .post("/api/run")
     .body(StringBody("[{\"id\":9,\"lat\":48.8601,
          \"lon\":2.3507,\"user\":\"lea\",\"timestamp\":1543775727}]")))
   .exec(http("find")
     .get("/api/run?user=lea&timestamp=1543775726,1543775729"))
   .exec(http("delete")
     .delete("/api/run/9"))

 setUp(
   scn.inject(atOnceUsers(5000))
 ).protocols(httpConf)
}
```

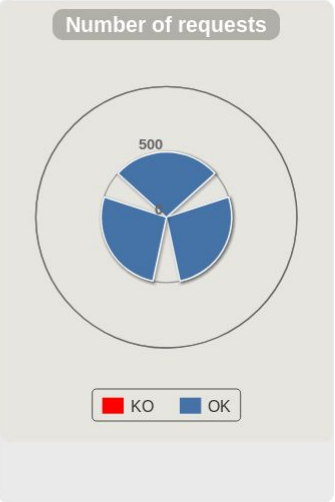
Notre profil de benchmark est de répéter plusieurs fois les requêtes suivantes : ajout d'une course, lecture d'une course et suppression d'une course. Nous avons effectué ce profil de trois façons différentes, le premier sur un petit nombre (500x), pour avoir des données lors d'un usage léger de Google Datastore et le second sur un nombre important (5000x) pour avoir des données lors d'un fort usage de Google Datastore et le dernier sur un usage minimale de Google Datastore (10x).

À des fins de comparaison, nous avons effectué une série de ping sur notre serveur et nous n'avons pas dépassé les 10 millisecondes :
--- tlcfitness.appspot.com ping statistics ---
707 packets transmitted, 707 received, 0% packet loss, time 1705ms
rtt min/avg/max/mdev = 7.625/8.051/10.084/0.307 ms

Rapport de benchmark

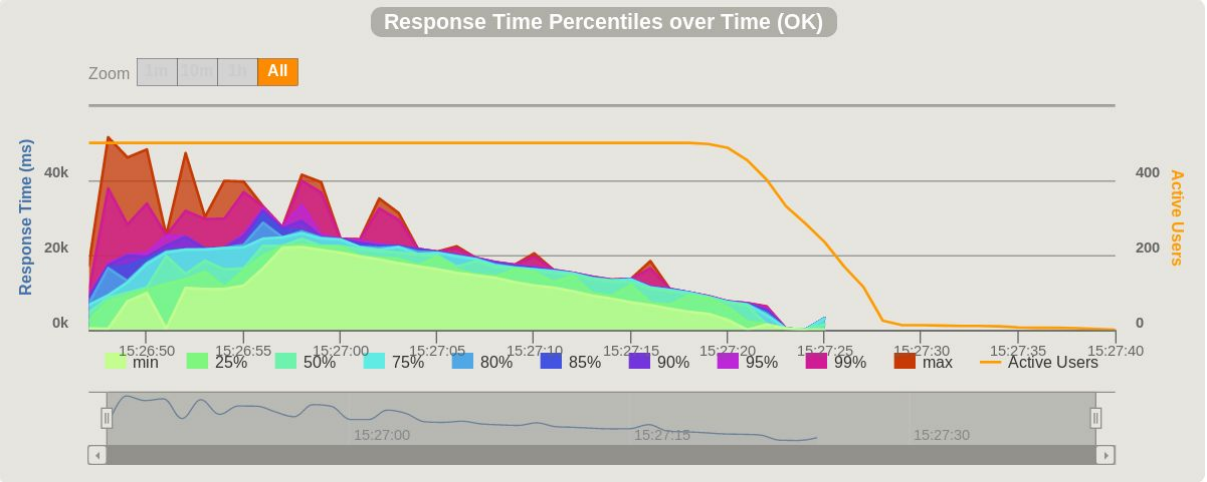
Profil de test à 500x



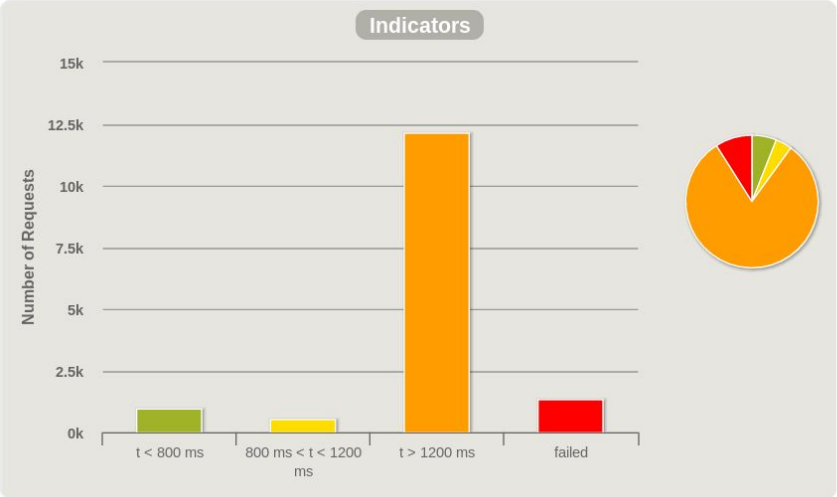


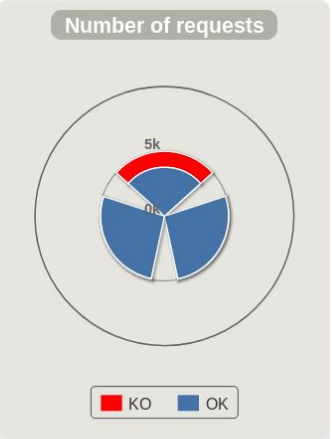
STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Req/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	1500	1500	0	0%	27.778	64	11388	19231	24246	31913	51533	12531	7697
add	500	500	0	0%	9.259	457	4652	6686	9297	10238	17286	4782	2605
find	500	500	0	0%	9.259	311	15728	21314	24501	27094	29114	15821	5467
delete	500	500	0	0%	9.259	64	16732	21342	25891	39881	51533	16990	7095



Profil de test à 5000x



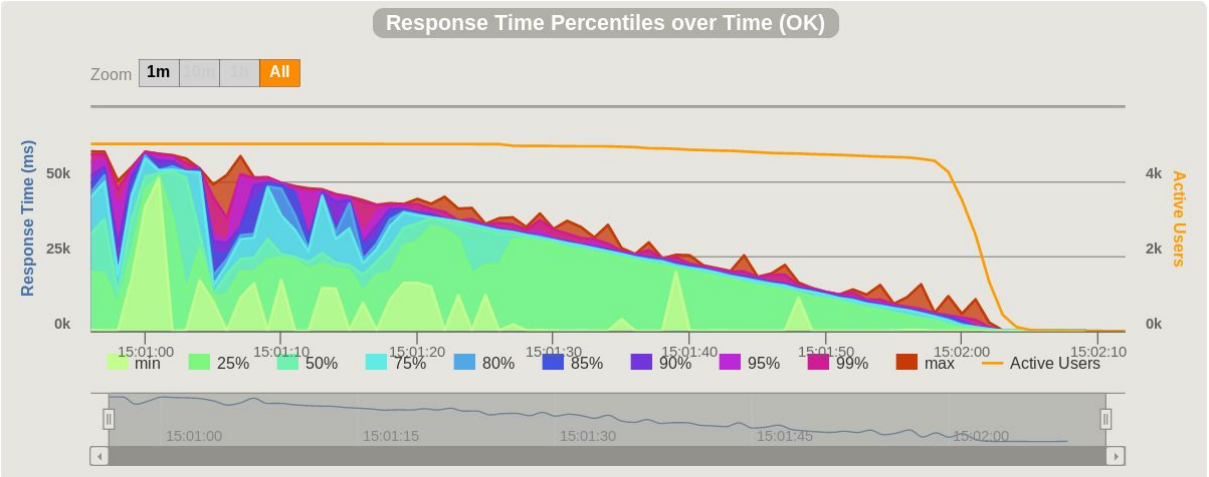


STATISTICS — Expand all groups | Collapse all groups

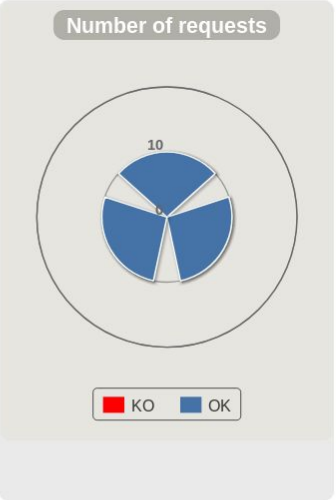
Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Req/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	15000	13648	1352	9%	194.805	51	19746	31320	52461	59098	60024	21075	15946
add	5000	3764	1236	25%	64.935	324	26750	44813	57438	60000	60024	28282	17932
find	5000	4950	50	1%	64.935	76	20943	28444	44699	53271	58424	21950	11338
delete	5000	4934	66	1%	64.935	51	4968	24619	36848	54739	60001	12994	13953

ERRORS

Error	Count	Percentage
i.n.c.ConnectTimeoutException: connection timed out: tlcfitness.appspot.com/216.58.198.212:443	1221	90.311 %
i.g.h.c.i.RequestTimeoutException: Request timeout to tlcfitness.appspot.com/216.58.198.212:443 after 60000 ms	118	8.728 %
status.find.in(200,201,202,203,204,205,206,207,208,209,304), found 500	7	0.518 %
j.n.s.SSLException: handshake timed out	6	0.444 %



Profil de test à 10x



STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Req/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	30	30	0	0%	10	63	557	660	1881	1883	1884	633	614
add	10	10	0	0%	3.333	577	1580	1880	1883	1884	1884	1303	595
find	10	10	0	0%	3.333	77	140	192	473	655	700	195	173
delete	10	10	0	0%	3.333	63	521	563	659	661	662	401	226

3. Analyse

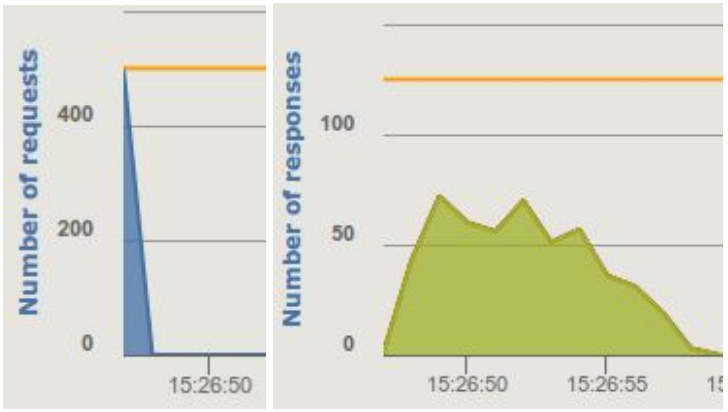
a. Première série de requêtes (500x)

Ce premier benchmark a nécessité 1 500 requêtes réparties entre la création, lecture et suppression de données pour une durée de 53 secondes.
L'ensemble des requêtes a été mené à son terme avec une moyenne d'environ 30 requêtes par seconde.
La requête la plus rapide a été traitée en 64 millisecondes alors que la plus lente a eu besoin de 51 533 millisecondes pour une moyenne de 12 531 millisecondes

Les requêtes de création sont plus rapides en moyenne (4 782 millisecondes) que celles de lecture et suppression (autour des 16 000 millisecondes)

Pendant la durée du benchmark 94 % des requêtes ont été traitées en plus de 1 200 millisecondes.
Le temps de réponse n'est donc pas très rapide mais le serveur et la base de données tiennent la charge et ne provoque aucune erreur.

L'intégralité des requêtes ont été envoyées en 1 secondes et les réponses correspondantes ont pour la plupart été obtenues en moins de 12 secondes d'attente.



b. Seconde série de requêtes (5000x)

Ce second benchmark a nécessité 15 000 requêtes tout en conservant le même profil que le premier benchmark. On peut observer que le nombre de requêtes est déjà plus élevé que lors du premier test puisque l'on monte à près de 200 requêtes par seconde.

On peut aussi observer que le temps de réponse est assez élevé puisqu'il dépasse assez facilement un temps de 20 secondes. Seul 6% des requêtes possèdent un temps de réponse normal, inférieur à 800 millisecondes.

La requête la plus rapide est de 51 millisecondes et se trouve être une requête de suppression. Les requêtes les plus lente correspondent aux requête d'insertion de donnée puisque le temps minimum pour effectuer ce type de requête s'élève à 324 millisecondes.

La différence la plus importante avec le premier benchmark est le nombre de requêtes qui n'ont pas pu répondre. Sur les 15 000 requêtes effectuées, 1 352 requêtes ont été en échec donc un taux d'erreur de 9% principalement répartie sur les requêtes d'insertion de données. Si l'on analyse le graphique du taux d'échec, on peut observer qu'un pic s'est trouvé au début du benchmark, on peut penser que c'est le moment choisis par Google pour augmenter les performances disponible pour notre serveur en raison de la forte demande.

c. Dernière série de requêtes (10x)

De la même manière que les précédents tests, nous avons effectué un dernier test avec uniquement 30 requêtes pour nous permettre de voir les temps de réponse que pourrait avoir une application avec une architecture en micro-service et un load balancing correct.

On peut donc observer que les temps de réponse sont déjà largement meilleur puisque 80% des requêtes répondent en moins de 800 millisecondes, que seul les requêtes d'insertion restent longues (une moyenne de 1.3 secondes) alors que la lecture et la suppression se limitent respectivement à 195 et 401 millisecondes.

4. Conclusion

Ce projet nous a permis de bien comprendre l'utilisation de Google Datastore et des services de Google Cloud en générale. Nous avons aussi appris à manipuler le framework Restlet pour créer des API REST. Les analyses nous ont aussi permis de comprendre les performances d'une solution en cloud.

Suite à nos différents bench, nous pouvons conclure que ce genre de solution ne peut pas être envisagé pour une application monolithique sur un serveur unique et possédant beaucoup de requêtes simultanées, seule une multitude de petits serveurs permettant de répartir les requêtes avec chacun un micro-service peut être intéressant et surtout performant.

Nous avons eu quelques difficultés pour développer les requêtes de recherche avec l'API de Google Datastore, notamment à cause des différentes implémentations de la librairie de Google Datastore en Java, qui n'ont pas l'air compatibles les unes avec les autres et qui sont pourtant détaillées dans les différents exemple fournis dans la documentation de Google.

Certains de notre groupe ont déjà manipulé d'autres outils d'hébergement cloud comme AWS. Leur conclusion est qu' AWS reste largement plus intuitif à configurer et à développer. Sur les services d'Amazon, la documentation est souvent plus complète et la communauté sur les différents forums plus prolifiques.