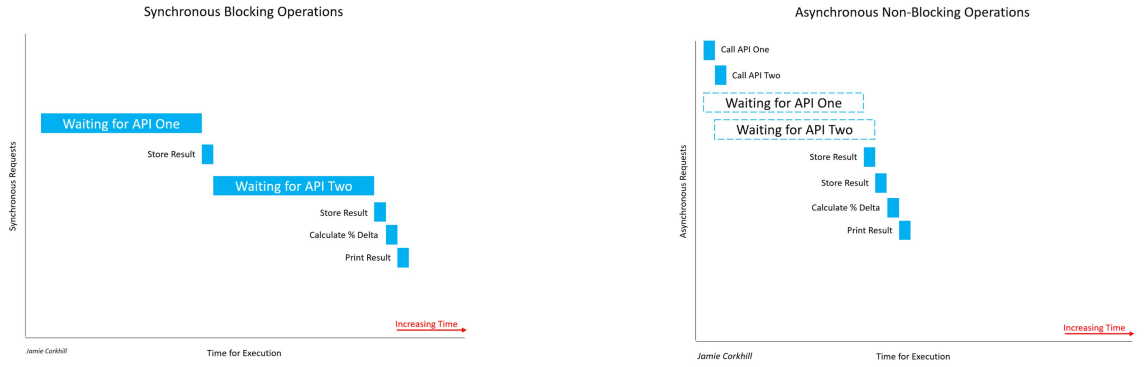
Etude NodeJS

NodeJS c'est quoi

Node JS est un langage interprété (javascript) qui est exécuté par un moteur très performant développé par Google : V8, lui écrit en C/C++. NodeJS profite donc des avantages des langages interprétés : langages haut niveau, pratiques, très rapide de développer des choses complexes, cross plateforme, environnement de développement simple.
Tout en limitant un peu le contrecoup sur les performance, grâce au performances du V8.

Un langage principalement asynchrone

En revanche NodeJS est d'avantage adapté à certains type de traitements.
il excel dans les traitements asynchrones. Chaque instance est monothread mais ne laisse jamais le thread dans un état d'attente bloquant. (grâce à l'event loop)



Dès qu'on sort de son domaine de prédilection pour faire des traitements lourds et ininterrompus (traitements à grosse charge CPU), on perd drastiquement en performance par rapport à un langage compilé comme C ou C++. (ex, traitement image, gros fichiers, vidéo etc…) C'est l'aspect monothread qui devient très limitant, en plus de performances moins bonnes.

Solutions aux limitations de NodeJS

Pour remédier en parti à ça il est possible de gérer des pool d'instances d'un même script nodeJS (ex pm2) qui va donc instancier plusieurs fois le même programme (donc plusieurs threads) et répartir la charge (les requêtes) entre les instances. Cependant ça ne s'apparente pas vraiment à ce qu'on pourrait faire en multithreading en C ou C++. Ici les programmes sont complètement indépendants. Dans le cas d'application stateless, ça ne pose aucun problème. Mais quand il faut commencer à se souvenir d'un état entre deux appels (comme les sessions en php par exemple) ou qu'il faut que toutes les instances partage ces données contextuelles (exemple socket.io et tous les sockets en cours), ça devient plus difficile. Il existe des solution, comme la mise en place d'une base de donnée temporaire, très rapide (NoSql) comme Redis, qui va jouer le rôle de mémoire partagée

entre les différentes instances. Cependant, plus on rajoute d'intermédiaires plus on impacte les performances.

Grâce à la N-API de NodeJS, qui permet l'interaction entres des programmes (modules) développées en C/C++, il est tout de même possible de réunir le meilleur des deux mondes. La partie NodeJS servira alors de point d'entrée du service, qui orchestrera/dispatchera les demandes vers des programmes natifs, multithreadés, et retournera le résultat à l'utilisateur quand la tâche sera complète, sans avoir interrompu (longtemps) son cours d'exécution.

Récap

Pour conclure, comme le décrit cet exemple, NodeJS s'adapte parfaitement à une architecture découpée en services ou microservices. Ou le nombre d'I/O est important. Il pourrait être très pertinent d'avoir des services en NodeJS devant les sources de données d'une architecture. Il conviendra cependant de concevoir son architecture en prenant en compte les ses contraintes. (découpage des tâches async/sync, préférer le stateless, ne pas chercher à tout faire systématiquement en NodeJS)

+	-
environnement de développement très simple à mettre en place	Peu performant pour les traitement à lourde charge CPU
très performant pour les applications orientées IO	Trop parfoi d'interdépendances entre les modules (npm)
Une multitude de librairies et de modules avec npm	versionning des modules (npm) ne respecte pas toujours la norme.
Api pour communiquer avec des programmes C/C++	monothread uniquement
très simple de développer une fois la notion d'asynchronisme assimilée	

Dans notre archi

Dans notre architecture il s'intègre parfaitement au poste de serveur de jeu et chat. En effet, avec la librairie socket.io il est extrêmement simple d'intégrer un serveur temps réel avec des interactions entre plusieurs utilisateurs. Quand viendra la question du dimensionnement il faudra comme évoqué dans cette analyse, mettre en place une solution pour assurer la communication entre les joueurs d'une même partie/ d'un même salon de tchat.
Toujours dans notre cas, on aurait aussi pu imaginer remplacer les services SpringBoot par des applications nodeJs. En effet, ces services ne font aucun traitements lourds et se trouvent entre différentes sources de données qui peuvent êtres traitées de manière asynchrone. (service de message, base de donnée, controleur REST). Il aurait même été

plus rapide à prendre en main que SpringBoot/Maven qui est assez difficile à aborder avec de simples connaissances Java pures. Le typages faibles que nous aurions perdu avec le javascript de NodeJs aurait alors pu être contrebalancé par l'utilisation de typescript qui l'aurait rendu notre code plus robuste plus robuste.

source :
schéma synchrone/asynchrones :
https://www.smashingmagazine.com/2019/02/node-api-http-es6-javascript/