Innovation, Sciences et Développement économique Canada
Office de la Propriété Intellectuelle du Canada

Innovation, Science and Economic Development Canada
Canadian Intellectual Property Office

CA 3188740 C 2023/08/29

(11)(21) 3 188 740

(12) BREVET CANADIEN
CANADIAN PATENT

(13) C

(86) Date de dépôt PCT/PCT Filing Date: 2022/03/14
(87) Date publication PCT/PCT Publication Date: 2023/05/05
(45) Date de délivrance/Issue Date: 2023/08/29
(85) Entrée phase nationale/National Entry: 2023/02/06
(86) N° demande PCT/PCT Application No.: CA 2022/050377

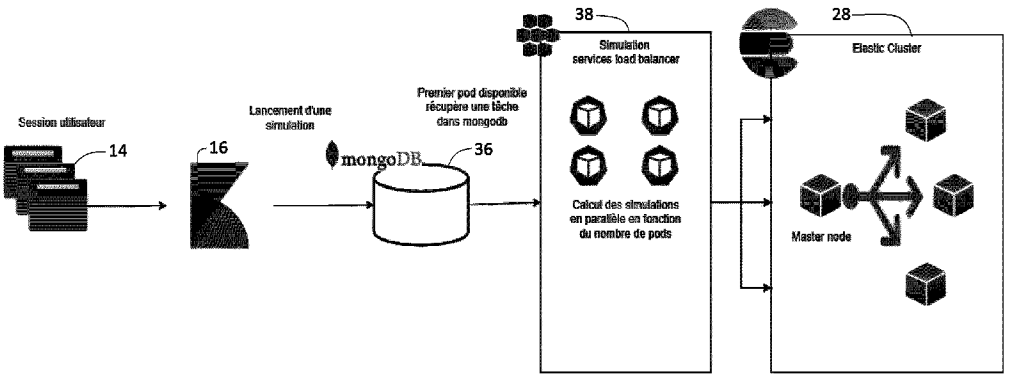
(51) Cl.Int./Int.Cl. *G06F 9/44* (2018.01), *G06F 16/903* (2019.01), *G06F 9/455* (2018.01)

(72) Inventeurs/Inventors:
LEPAGE, ALAIN, CA;
VINCENT, PASCAL, CA

(73) Propriétaire/Owner:
SERVICES MAKILA INC., CA

(74) Agent: LAVERY, DE BILLY, LLP

(54) Titre : SYSTEME ET PROCEDE D'OPTIMISATION D'UN OUTIL DE SIMULATION
(54) Title: SYSTEM AND PROCESS FOR OPTIMIZING A SIMULATION TOOL



(57) Abrégé/Abstract:
On vise a intégrer une interface applicative de simulation de données dans un tableau de bord délivré par un plugin Kibanamc. Cette capacité supplémentaire ajoutée a la suite ELK[MC] nécessite des composantes additionnelles : L'interface applicative (API) de simulation en Python et le système de file d'attente des tâches (Python[MC], NodeJs[MC], MongoDB[MC]) et l'intégration d'un nombre important de données a cheque simulation.

ABRÉGÉ

On vise à intégrer une interface applicative de simulation de données dans un tableau de bord délivré par un plugin Kibana[MC]. Cette capacité supplémentaire ajoutée à la suite ELK[MC] nécessite des composantes additionnelles : L'interface applicative (API) de simulation en Python et le système de file d'attente des tâches (Python[MC], NodeJs[MC], MongoDB[MC]) et l'intégration d'un nombre important de données à chaque simulation.

Date Reçue/Date Received 2023-02-06

TITRE DE L'INVENTION

SYSTÈME ET PROCÉDÉ D'OPTIMISATION D'UN OUTIL DE SIMULATION

DOMAINE DE L'INVENTION

[0001]La présente invention concerne le domaine du traitement d'informations à grande échelle et, en particulier, à un système et un procédé d'optimisation d'un outil de simulation.

CONTEXTE DE L'INVENTION

[0002]Elasticsearch[MC] est un moteur de recherche et d'analyse distribué, gratuit et ouvert pour tous les types de données, y compris textuelles, numériques, géospatiales, structurées et non structurées. Elasticsearch[MC] est basé sur Apache Lucene[MC] et a été publié pour la première fois en 2010 par Elasticsearch N.V. (maintenant connu sous le nom d'Elastic). Connu pour ses API REST (API : « application programming interface » ou interface de programmation applicative; REST : « Representational State Transfer » i.e. un style d'architecture caractéristique des programmes qui s'appuient sur les propriétés inhérentes des hypermédias et du protocole HTTP pour créer et modifier l'état d'un objet accessible via une URL), sa nature distribuée, sa vitesse et son évolutivité, Elasticsearch[MC] est le composant central de Elastic Stack[MC], un ensemble d'outils gratuits et ouverts pour l'ingestion, l'enrichissement, le stockage, l'analyse et la visualisation des données. Communément appelée ELK Stack[MC] (ELK : Elasticsearch[MC], Logstash[MC] et Kibana[MC]), Elastic Stack[MC] comprend une collection d'agents d'expédition légers connus sous le nom de Beats[MC] pour envoyer des données à Elasticsearch[MC].

[0003]Kibana[MC] est une application « front-end » gratuite et ouverte qui se trouve au-dessus de Elastic Stack[MC], offrant des capacités de recherche et de visualisation des données pour les données indexées dans Elasticsearch[MC]. Kibana[MC] sert également d'interface utilisateur pour la surveillance, la gestion et la sécurisation d'un cluster Elastic Stack[MC], ainsi que le hub centralisé pour les solutions intégrées développées sur la Elastic Stack[MC]. Kibana[MC] permet la recherche, l'affichage et la visualisation des données indexées dans

Date Reçue/Date Received 2023-02-06

Elasticsearch[MC] et l'analyse des données via la création de graphiques à barres, de camemberts, de tableaux, d'histogrammes et de cartes. Une vue de tableau de bord combine ces éléments visuels pour ensuite être partagés via un navigateur pour fournir des vues analytiques en temps réel dans de grands volumes de données à l'appui de cas d'utilisation tels que le risque d'entreprise et l'analyse décisionnelle.

[0004] R[MC] est un environnement logiciel libre utilisé pour le calcul statistique et les graphiques. Il compile et s'exécute sur une grande variété de plates-formes UNIX[MC], Windows[MC] et MacOS[MC]. R[MC] fournit une grande variété de techniques statistiques (modélisation linéaire et non linéaire, tests statistiques classiques, analyse de séries chronologiques, classification, clustering, ...) et graphiques, et est hautement extensible.

[0005] Plumber[MC] est un système d'opération qui permet de créer une API Web en décorant un code source R[MC] existant avec des commentaires de type roxygen2.

[0006] La taille et la complexité sans cesse croissantes des ensembles de données scientifiques modernes défient constamment les capacités des méthodes de calcul statistique existantes. Le calcul parallèle statistique haute performance est une solution prometteuse pour relever ces défis. Cependant, les techniques de calcul statistique parallèle introduisent de nombreuses complexités de mise en œuvre, entraînant un besoin de processus plus efficaces et rationalisés. En réponse à ce besoin, il est connu d'utiliser pR[MC], un middleware léger et facile à utiliser pour le R-Engine riche en statistiques qui permet en outre la parallélisation dans le calcul statistique.

[0007] Node.Js[MC] est un environnement d'exécution JavaScript open source, multiplateforme et « back-end » qui s'exécute sur un moteur et exécute du code JavaScript en dehors d'un navigateur Web.

[0008] MongoDB[MC] est un programme de base de données multiplateforme orienté document disponible en source. Classé comme programme de base de données NoSQL, MongoDB[MC] utilise des documents de type JSON[MC] avec des schémas facultatifs.

Date Reçue/Date Received 2023-02-06

[0009] Les Pods Kubernetes [MC] sont les objets déployables les plus petits et les plus basiques de Kubernetes[MC]. Un Pod représente une instance unique d'un processus en cours d'exécution dans un cluster.

[0010] On connaît dans le domaine le brevet US10803401B2 (HAMMOND), de même que les demandes de brevets US20210064346A1 (EXERTIER), US20180357047A1 (BROWN) et WO2020136680A1 (SABHARWAL).

OBJETS DE L'INVENTION

[0011] Selon des réalisations préférentielles de l'invention, on vise à intégrer une interface applicative de simulation de données dans un tableau de bord délivré par un plugin Kibana[MC]. Cette capacité supplémentaire ajoutée à la suite ELK[MC] nécessite des composantes additionnelles : L'interface applicative (API) de simulation en Python[MC] et le système de file d'attente des tâches (Python[MC], NodeJs[MC], MongoDB[MC]) et l'intégration d'un nombre important de données à chaque simulation. Par exemple, on peut utiliser plus de 8 millions de données pour simuler une grande entreprise pendant 11 ans chaque mois.

[0012] Selon une réalisation préférentielle de l'invention, Kibana Server[MC], Node.Js[MC] et R-Engine[MC] ou Python[MC] sont dans un même Pod Kubernetes[MC]. Du fait des spécificités du R-Engine[MC] [1], on fait face à une complexité dans la communication (par ligne de commande dynamique) et dans les retours d'erreur entre l'API Kibana[MC] et R-Engine[MC]. De plus pour certains environnements cibles, lorsque R-Engine et Kibana-Server sont dans le même Pod kubernetes, cela introduit une vulnérabilité.

[0013] Selon une réalisation préférentielle, on propose un procédé de simulation comprenant des étapes de:

réception d'une première demande de simulation d'un agent par une interface applicative (API) et création d'une tâche de simulation;

vérification par un premier service si la demande de simulation est une tâche principale,

si oui, le premier service crée des sous tâches par entités pour effectuer la simulation en simultanée et un calcul de la simulation;

sinon, si la tâche est une sous tâche, on calcule une simulation pour une entité correspondante;

Date Reçue/Date Received 2023-02-06

indexation de la simulation dans un deuxième service;

mise à jour du statut de la simulation par le premier service;

compilation d'un résultat de génération de langage naturel dans le premier service;

interrogation du résultat de la simulation de l'agent par l'interface applicative (API) à intervalle régulier jusqu'à ce que la simulation soit complétée;

requête du résultat de la simulation au deuxième service; et

transmission du résultat de simulation à l'interface applicative.

[0014] Selon des réalisations préférentielles, le premier service et deuxième service sont installés dans un cluster Kubernetes[MC]. L'interface applicative (API) peut être un plugin Kibana[MC] qui est installé sur un Pod d'un cluster Kubernetes[MC]. Le premier service peut comprendre Python[MC] et la librairie cliente MongoDB[MC] dans un Pod scalable sur un cluster Kubernetes[MC]. Le deuxième service (28) peut comprendre Elasticsearch[MC] installé sur un cluster de machine virtuelle ou un cluster Kubernetes[MC].

[0015] Selon une réalisation préférentielle, on propose également un système de simulation comprenant :

une interface applicative (API) configurée pour recevoir une première demande de simulation d'un utilisateur via un agent;

une file d'attente de travail installée dans un Pod d'un cluster Kubernetes[MC] pour recevoir la première demande de simulation de l'utilisateur pour répartir une charge de travail sur différentes instance d'un premier service; et

un engin analytique installé dans le Pod du cluster Kubernetes[MC] pour recevoir des résultats de la simulation et le transmettre à l'interface applicative (API).

[0016] D'autres objets, avantages et fonctions de la présente invention deviendront plus apparentes lors de la description suivante de modes de réalisations possibles, donnés à titre d'exemples seulement, en relation aux figures suivantes.

Date Reçue/Date Received 2023-02-06

BRÈVE DESCRIPTION DES FIGURES

[0017]La Figure 1 est un diagramme schématique d'un système représentant un premier prototype, selon une réalisation préférentielle de la présente invention.

[0018]La Figure 2 est un diagramme schématique montrant un procédé d'opération du premier prototype, selon une réalisation préférentielle de la présente invention.

[0019]La Figure 3 est un diagramme schématique d'un système représentant un deuxième prototype, selon une réalisation préférentielle de la présente invention

[0020]La Figure 4 est un diagramme schématique montrant un système, selon une réalisation préférentielle de la présente invention.

[0021]La Figure 5 est un diagramme schématique montrant un procédé d'opération, selon une réalisation préférentielle de la présente invention.

[0022]La Figure 6 est un diagramme schématique montrant un système, selon une réalisation préférentielle de la présente invention.

DESCRIPTION DES MODES DE RÉALISATIONS DE L'INVENTION

[0023]La présente invention est illustrée plus en détail par les exemples non limitatifs suivants.

[0024]Comme l'appréciera la personne du métier à la lecture de la description suivante, divers aspects décrits ici peuvent être mis en œuvre sous la forme d'un dispositif, d'un procédé ou d'un produit de programme informatique (par exemple, un support ou mémoire lisible par ordinateur non transitoire ayant des instructions exécutables par ordinateur qui, une fois exécutées, exécutent les opérations ou étapes). Ainsi, ces aspects peuvent prendre la forme d'un mode de réalisation entièrement matériel, d'un mode de réalisation entièrement logiciel ou d'un mode de réalisation combinant des aspects logiciels et matériels.

[0025]En outre, de tels aspects peuvent prendre la forme d'un produit de programme informatique stocké par un ou plusieurs supports de stockage

Date Reçue/Date Received 2023-02-06

lisibles par ordinateur ayant un code de programme lisible par ordinateur, ou des instructions, incorporés dans ou sur les supports de stockage. Tout support de stockage lisible par ordinateur approprié peut être utilisé, y compris les disques durs, les CD-ROM, les dispositifs de stockage optiques, les dispositifs de stockage magnétiques et/ou toute combinaison de ceux-ci.

[0026] Selon une réalisation préférentielle de l'invention, Kibana Server^{MC}, Node.Js^{MC} et R-Engine^{MC} ou Python^{MC} et tous autres engins analytiques sont dans un même Pod Kubernetes^{MC}. Du fait des spécificités du R-Engine^{MC} [1], les inventeurs ont fait face à une complexité dans la communication (par ligne de commande dynamique) et dans les retours d'erreur entre l'API Kibana^{MC} et R-Engine^{MC}. De plus pour certains environnements cibles, lorsque R-Engine^{MC} et Kibana-Server^{MC} sont dans un même Pod Kubernetes^{MC}, cela introduit une vulnérabilité.

[0027] Selon une autre réalisation préférentielle de l'invention, on fait résider R-Engine^{MC} séparément et on invoque ce dernier par API REST. Dans ce sens, l'infrastructure « Backend » comprend : Kibana^{MC} pour la visualisation; ElasticSearch^{MC} pour la persistance; et un mécanisme ou moyens (PLUMBER^{MC}) pour transformer les fonctions R en API REST et construire le serveur http.

[0028] Le PLUMBER^{MC} identifie et achemine les requêtes http pour invoquer les services du R-Engine^{MC}. Dans le serveur Kibana^{MC}, un service invoque l'API REST et avec le retour, met à jour ElasticSearch et construit la réponse retournée par l'API "Kibana^{MC} serveur" au client pour actualiser son tableau de bord avec le scénario demandé. Contrairement au cas standard, les clients d'un tableau de bord ne font pas que le visualiser, ils influent sur sa visualisation via la simulation de scénario.

[0029] Il faut prendre en considération que plusieurs clients d'un même tableau de bord invoquent concurremment les services R-Engine^{MC}. D'après une réalisation préférentielle de l'invention, on doit donc lancer plusieurs threads (un par simulation), ce qui nativement est compromis d'une part par le Threading collaboratif de R-Engine^{MC} et d'autre part, par le côté single thread http du

Date Reçue/Date Received 2023-02-06

serveur construit par PLUMBER[MC]. Or ce dernier ne peut pas interpréter plus d'une requête http à la fois.

[0030] Ces limites technologiques rendent incertaines la disponibilité et la consistance des visualisations, conséquence d'un double risque de traitement séquentiel des scénarios par R-Engine[MC] et PLUMBER[MC]. Le traitement séquentiel occasionne des expirations de délai (time out) et une inconsistance des tableaux de bord qui ne refléteront plus le scénario de l'utilisateur pour cause d'interprétations erronées.

[0031] Plus précisément, si les scénarios sont logés en mémoire par R-ENGINE[MC], tout traitement sur ces scénarios sera atomique : R-ENGINE[MC] ne redonnera pas la main aux processus concurrents tant que le scénario ne sera pas construit. Cela est particulièrement dommageable pour les scénarios très courts, qui restent dans l'attente inutilement de scénarios plus longs. Le fait que PLUMBER[MC] soit entrelacé dans les scripts de R[MC], fait que les deux limites technologiques "single http" de PLUMBER[MC] et "single threaded" du R-ENGINE[MC] ne peuvent pas être abordées séparément.

[0032] Dans ce sens, traiter la problématique "single http" de PLUMBER[MC] laisse une forte incertitude effective pour les scénarios CPU intensive, alors que lancer plusieurs instances du R-Engine[MC] nécessite de gérer le goulot d'étranglement posé par le côté single http de PLUMBER[MC]. Une autre problématique s'ajoute alors : la limite de la capacité de traitement des instances de R[MC], suite aux appels http. Il faut donc parvenir à déterminer une composante de balancement pour garantir un taux d'utilisation acceptable (>90 %) de chaque instance du R-Engine[MC].

[0033] La parallélisation des processus en Machine Learning, et dans notre cas en simulation, fait actuellement l'objet d'études de R&D sur différents aspects comme nous pouvons le constater dans la littérature [2]. Nous avons dû par conséquent développer une nouvelle méthode de parallélisation afin de tenter de répondre à nos contraintes.

[0034] Nous pouvons voir dans la littérature que la parallélisation des traitements d'indexation et bulk ElasticSearch [3][4] est un sujet traité via des plateformes de répartiteur de charges (Load Balancer) génériques. Ceux-ci

Date Reçue/Date Received 2023-02-06

demandent une intégration lourde en termes d'infrastructure ainsi que de coût de licence et maintenance, ce qui amène une complexité avec un surdimensionnement qui n'est pas utile dans plusieurs cas d'utilisation.

[0035]Les solutions existantes ne règlent qu'un seul des deux obstacles à surmonter. Leur application indépendante dans notre contexte s'expose à un mauvais taux d'utilisation des instances du R-ENGINE^{MC}.

[0036]D'après des réalisations préférentielles de l'invention, on fait collaborer des scripts R-ENGINE^{MC} avec un moteur d'indexation ElasticSearch^{MC}. Ceci permet à un utilisateur de simuler et d'intégrer des scénarios pour réaliser une analyse des résultats sur plusieurs années.

[0037]Au lieu d'utiliser un process réalisant plusieurs traitements, une réalisation préférentielle de la présente invention utilise plusieurs process traitant en parallèle. Pour faire face à la chute du taux d'utilisation, nous avons enrichi notre hypothèse afin que chaque simulation dans le pool devienne un consommateur indépendant des autres. Ainsi la structure se transforme en une liste FIFO (First In First Out), chaque simulation dans le pool qui finit va chercher une requête http de la liste FIFO. Dans cette implémentation, la consommation des simulations se fait par appel récursif. L'analyse des logs pour un banc d'essai de plusieurs lots de 500 requêtes http, de durée aléatoire, avec 2,8,16 et 32 instances du R-ENGINE^{MC}, montrait qu'en tout temps le nombre de tâches asynchrones en cours était égal au nombre d'instances de R-ENGINE^{MC}.

[0038]Pour optimiser ceci, il nous a fallu décomposer le code des simulations en éléments de base assez petits pour obtenir des appels en parallèle sans la présence d'un gros élément qui aurait généré de l'attente, tout en gardant un niveau de granularité assez élevé pour que le temps ne soit pas exclusivement pris en appel de procédure entre NodeJS^{MC}, PLUMBER^{MC} et R^{MC}.

[0039]L'invocation des services R^{MC} par API REST s'est trouvée compromise par le caractère single http du serveur qui prend en charge les API. Ni le multiplexage par fonction ou par port n'ont permis de lever cet obstacle, s'étant traduit respectivement soit par un refus de connexion ou un traitement séquentiel des API. Quant à l'obstacle posé par le threading collaboratif, nos

Date Reçue/Date Received 2023-02-06

tâches étant CPU intensive, et de surcroit, logées en mémoire, devenaient des opérations atomiques : ce qui est rédhibitoire pour de courtes simulations.

[0040] Le lancement de plusieurs instances résultait en un taux d'utilisation des instances imprévisibles et des techniques d'affectation ne pouvant s'appliquer qu'à deux instances. Nous avons réalisé un avancement technologique, un répartiteur de charge (Load Balancer) spécialement adapté aux cas où l'on peut mettre une relation d'un pour un entre l'instance de serveur http et l'instance de service invoquée en les faisant tourner dans le même process.

[0041] Cet avancement s'appuie sur la notion de pool de promesses, une notion inexistante dans Node.Js[MC] que nous avons dû développer. Ce pool prend la forme d'un tableau de promesses qui récursivement, en exploitant la capacité de chainage de promesses, va chercher dans une file FIFO la prochaine requête à exécuter.

[0042] Cet avancement est reproductible et pourra s'appliquer pour tout Framework construisant les API REST directement à partir du script via des décorateurs. Comme expérimenté, de tels serveurs sont incapables d'écouter plusieurs requêtes http : ce qui amène un goulot d'étranglement et nuit à la concurrence. Sans cet avancement, nous n'aurions pas pu implémenter notre scénario de simulations multiples.

[0043] Cette solution nous permet d'obtenir un résultat en 5 minutes, ce qui est acceptable mais qui n'atteint pas un objectif de 3 minutes qui serait souhaitable. Afin d'améliorer les temps d'indexation de données, on peut utiliser Python[MC] pour réaliser les bulks d'indexation des 8 millions de lignes. En passant les scripts de R-Engine[MC] à Python[MC] il est possible de garantir une meilleure intégration technique globale.

[0044] Afin de rendre dynamiques les visualisations de Kibana via la simulation de scénarios, selon une réalisation préférentielle, on rassemble Kibana[MC], Node.js[MC] et R-Engine[MC] sur le même cluster Kubernetes[MC] dans l'architecture. La communication par ligne de commande s'est révélée non générique, avec une gestion d'erreurs peu robuste entre Kibana et R-Engine. Il est donc préférable de faire résider le R-ENGINE[MC] dans un Pod Kubernetes[MC] différent et d'invoquer ses services par appel http REST. Pour construire l'infrastructure

Date Reçue/Date Received 2023-02-06

REST, PLUMBER^{MC} construit le serveur qui écoute les appels REST et invoque les fonctions d'un script, via les décorations dans le script R^{MC}. Cependant, la nature CPU intensive des scénarios est incompatible avec l'aspect single threaded de R-Engine^{MC}.

[0045] Nous avons donc réalisé une première étude préliminaire et construit un banc de test de plusieurs usagers (robots) lançant concurremment via Kibana^{MC} les scénarios usuels avec différentes tailles et complexités. Nous avons ainsi mesuré la durée de complétion des scénarios.

[0046] En référence à la Figure 2, un premier prototype implique plusieurs processus : Kibana^{MC}, ElasticSearch^{MC}, PLUMBER^{MC} pour construire le serveur adossé au R-Engine^{MC}. Dans ce procédé, quand un utilisateur 10 demande une simulation 12 via un agent 14, l'appel REST est soumis à Kibana^{MC}, soit une simulation API 16, qui loge un second appel http, soit un POST simulation 18, vers le serveur 20 généré par PLUMBER^{MC}. Ce dernier serveur 20 invoque, via une demande de calcul de simulation 22, les fonctions 24 du R-Engine^{MC} pour effectuer la simulation et retourne les données de simulation, soit l'indexation de la simulation 26, qui serviront à la mise à jour ElasticSearch^{MC} dans Elastic^{MC} et à l'API Kibana^{MC} pour les relayer à l'utilisateur 30, 32.

[0047] Nos essais ont cependant montré que tant qu'une des URL vers la fonction est en cours, invoquer la même fonction avec la seconde URL générait une erreur de connexion.

[0048] Nous avons donc dû travailler à l'intégration d'un répartiteur de charge (« load balancer ») et réaliser de nouveaux travaux de prototypage. Il existe une limitation entre PLUMBER^{MC} et R^{MC} : si une simulation est en cours, alors aucune nouvelle demande ne peut passer. A ce stade il était impossible de déterminer si le problème venait de PLUMBER^{MC} n'étant pas capable d'écouter plus qu'une connexion http.

[0049] Nous avons par la suite initié la construction de deux serveurs PLUMBER^{MC} sur deux ports différents. Les essais ont validé le concept, le lancement simultané des deux requêtes était accepté. Cependant l'examen des logs d'exécution montrait qu'une des simulations devait attendre que l'autre finisse avant de débuter, ce qui était rédhibitoire.

Date Reçue/Date Received 2023-02-06

[0050] En effet notre plateforme multi-utilisateur et multi-client permet d'obtenir des temps optimaux sans démultiplier l'infrastructure, ce qui réduit les coûts de location et de maintenance. De ce fait nous ne pouvons accepter une solution ou un utilisateur/client attendrait que la simulation d'un autre client soit terminée avant que la sienne commence.

[0051] Une analyse approfondie de nos résultats nous a conduits à identifier le risque de deux obstacles imbriqués : l'aspect single thread http de PLUMBER^{MC} (que nous venions de résoudre par répartition de charge ou « load balancing ») et le caractère « single threaded » du R-ENGINE^{MC}. Plus précisément, les données de simulation sont chargées entièrement en mémoire par R-ENGINE^{MC}. Les opérations en mémoire étant atomiques, leur traitement ne peut pas être interrompu : ce qui bloque les autres traitements concurrents. Les courtes simulations sont alors pénalisées car elles attendent la fin des plus longues.

[0052] Nous avons revu nos hypothèses en conséquence, pour que les instances de serveur PLUMBER^{MC} (par port) s'exécutent sur des processus séparés. Cette hypothèse part du principe que le script R^{MC} s'exécute sur le même process que le serveur PLUMBER^{MC} qui l'a invoqué. Donc indirectement, il y aura autant d'instances du R-ENGINE^{MC} que de serveurs PLUMBER^{MC}. Les process s'exécutent soit en parallèle sur des processeurs séparés et/ou en multithreading avec une isolation mémoire, ce qui garantit la concurrence.

[0053] Pour atteindre nos objectifs, nous avons de nouveau revu notre approche afin que la construction des serveurs PLUMBER^{MC}/port s'effectue dans des processus différents. Nous avons construit un nouveau banc de test, avec deux instances lançant deux serveurs PLUMBER^{MC}, écoutant sur la même adresse IP et sur deux ports différents.

[0054] Notre test consistait à lancer en même temps deux simulations de durée 30s et 15s et évaluer si le temps d'exécution était inférieur à leur somme (45s). Les simulations se sont terminées en 32s, ce qui montrait que nous étions parvenus à dépasser l'obstacle du single http de PLUMBER^{MC} et du « single threaded » du R-ENGINE^{MC}.

Date Reçue/Date Received 2023-02-06

[0055] Grâce à ce test, nous pouvons confirmer que les tâches se lancent bien en parallèle et que les résultats obtenus donnent des accès parallèles.

[0056] En référence aux Figures 3 à 6, un deuxième prototype implique plusieurs processus : Kibana[MC], ElasticSearch[MC], MongoDB[MC] pour construire les Pod Kubernetes[MC] 11 adossé au R-Engine[MC] avec Python[MC] Le diagramme représente les interactions entre les différents acteurs (clients, services ou persistance de données) ainsi la séquence chronologique des différents évènements.

[0057] Le client ou utilisateur 10 via l'agent 14 démarre une nouvelle simulation 12, une nouvelle tâche 34 est créée par l'Api de simulation 16. Le service Python[MC] 38 va prendre la prochaine tâche non traitée 35 et l'exécuter. Le service Python[MC] vérifie en effectuant une requête 40 auprès du service MongoDB[MC] 36 s'il y a une nouvelle tâche. Cette vérification fait partie d'un processus qui est répété dans le bloc 37.

[0058] Le service Python 38 vérifie dans le bloc 39 le type de la tâche 40 : s'il s'agit d'un type de tâche principale 42, le service Python 38 distribue le traitement de la tâche en créant une sous tâche 44 par entité (une entité est un ensemble de données sur lesquels une simulation peut être appliquée de façon cohérente), sinon le service effectue le calcul de la simulation 46 pour l'entité. Une simulation peut être exécutée en parallèle en fonction du nombre d'instances du service Python[MC] et du nombre d'entités. Dès qu'une sous tâche 44 est complétée, le résultat de la simulation est indexé 48 dans Elastic[MC] 28. Lorsque toutes les sous tâches d'une simulation sont complétées : le statut de la tâche principale est mis à jour 50 et le résultat de génération de langage naturel ou « Natural Laguage Generation : NLG » 52 de chaque entité est compilé afin un résultat agrégé sur l'ensemble des entités traitées.

[0059] Pendant ce temps, l'agent 14 requête le résultat de la simulation 54 à l'API de simulation 16 à intervalle régulier jusqu'à ce que la simulation soit complétée. L'API de simulation 16 requête le résultat de la simulation 56 au deuxième service 28, soit à Elastic[MC]. Le deuxième service 28 transmet ensuite le résultat de la simulation 58 à l'API de simulation 16.

Date Reçue/Date Received 2023-02-06

[0060] Dans ce procédé, quand un utilisateur 10 demande une simulation 12 via un agent 14, l'appel REST est soumis à Kibana^{MC}, soit une simulation API 16, qui loge un second appel http, soit une nouvelle tâche de simulation 34, vers le serveur 36 généré par MongoDB^{MC}.

[0061] Classiquement, ce cas de figure est adressé par l'ajout de la requête http dans une file que les serveurs PLUMBER^{MC} consomment. Chaque requête dans ce sens a un hash construit à partir de : ID Simulation, ID Tableau de bord et ID client. Ce hash évite de refaire une simulation. De plus, combiné avec la notion de favori, il permet d'anticiper pour générer à l'avance les simulations pendant les périodes creuses. Notre problématique réside ici au niveau de l'utilisateur : comment assigner chaque requête à un serveur PLUMBER^{MC} tout en garantissant un taux d'utilisation des instances du R-ENGINE^{MC} supérieur à 80 %.

[0062] Selon des réalisations préférentielles de l'invention, le deuxième prototype s'adressant à des entreprises de grande taille, l'utilisation des instances doit être proche de 100%. Non seulement pour pouvoir scaler ou étendre à plus de deux instances sans perte de taux d'utilisation, mais aussi pour éviter les problèmes de timeout liés à l'utilisation d'écoute sur des ports http, qui est naturellement limitée en temps.

[0063] En effet, pour une simulation, les tests se décomposent sur le lancement de plusieurs processus en parallèle. Il faut attendre que chacun des processus ait répondu pour obtenir une réponse globale. Donc lorsqu'un processus met beaucoup plus de temps que les autres pour répondre, nous observons une attente de tous les autres processus avant de terminer globalement. Ceci diminue drastiquement le taux d'utilisation dans notre cas, car il n'est pas possible d'identifier à l'avance les processus les plus longs et donc de les répartir sur les différentes instances Python^{MC}.

[0064] La parallélisation des processus en apprentissage automatique ou « Machine Learning », et dans notre cas en simulation en simultanée, fait actuellement l'objet d'études de R&D (Recherche et Développement) sur différents aspects, comme nous pouvons le constater dans la littérature. Nous avons par conséquent développé une méthode de parallélisation afin de tenter

Date Reçue/Date Received 2023-02-06

de répondre à nos contraintes. La parallélisation doit permettre d'effectuer plusieurs tâches de « Machine Learning » en même temps, en prenant en compte la prédiction du temps requis pour chaque tâche de « Machine Learning » afin de répartir ces tâches sur plusieurs processus Python 38. Le but étant de répartir le plus possible l'utilisation de tous les POD Kubernetes Python^{MC} 24 selon un critère de temps de simulation prévue et non pas selon un nombre de simulation, afin que le temps de la simulation globale soit le plus court possible.

[0065]Les revendications ne doivent pas être limitées dans leur portée par les réalisations préférentielles illustrées dans les exemples, mais doivent recevoir l'interprétation la plus large qui soit conforme à la description dans son ensemble.

RÉFÉRENCES

[1] SHAH, Neil, KORA, Guruprasad, BREIMYER, Paul, et al. pR: Enabling Automatic Parallelization of Data-Parallel Tasks and Interfacing Parallel Computing Libraries in R with Application to Fusion Reaction Simulations. The, 2010. http://nshah.net/publications/pR_applications.useR.2010.pdf

[2] Memeti, S., Pllana, S., Binotto, A. et al. Using meta-heuristics and machine learning for software optimization of parallel computing systems: a systematic literature review. Computing 101, 893–936 (2019). https://doi.org/10.1007/s00607-018-0614-9

[3] Li, Y.; Jiang, Y.; Gu, J.; Lu, M.; Yu, M.; Armstrong, E.M.; Huang, T.; Moroni, D.; McGibbney, L.J.; Frank, G.; Yang, C. A Cloud-Based Framework for Large-Scale Log Mining through Apache Spark and Elasticsearch. Appl. Sci. 2019, 9, 1114. https://doi.org/10.3390/app9061114

[4] Kumar P., Kumar P., Zaidi N., Rathore V.S. (2018) Analysis and Comparative Exploration of Elastic Search, MongoDB and Hadoop Big Data Processing. In: Pant M., Ray K., Sharma T., Rawat S., Bandyopadhyay A. (eds) Soft Computing: Theories and Applications. Advances in Intelligent Systems and Computing, vol 584. Springer, Singapore. https://doi.org/10.1007/978-981-10-5699-4_57

Date Reçue/Date Received 2023-02-06

REVENDICATIONS

1. Procédé de simulation comprenant des étapes de:

réception d'une première demande de simulation (12) d'un agent (14) par une interface applicative (16) et création d'une tâche de simulation (34);

vérification par un premier service (38) si la demande de simulation (12) est une tâche principale (42),

si oui, le premier service (38) crée des sous tâches (44) par entités pour effectuer la simulation en simultanée et un calcul de la simulation;

sinon, si la tâche (34) est une sous tâche, on calcule (46) la simulation pour une entité correspondante;

indexation (48) de la simulation dans un deuxième service (28);

mise à jour (50) du statut de la simulation par le premier service (38);

compilation d'un résultat de génération de langage naturel (52) dans le premier service (38);

interrogation (54) du résultat de la simulation de l'agent (14) par l'interface applicative (16) à intervalle régulier jusqu'à ce que la simulation soit complétée;

requête (56) du résultat de la simulation au deuxième service (28); et

transmission (58) du résultat de simulation à l'interface applicative (16).

2. Le procédé selon la revendication 1, dans lequel le premier service (38) et deuxième service (28) sont installés dans un cluster Kubernetes (11).

3. Le procédé selon la revendication 1, dans lequel l'interface applicative (16) est un plugin Kibana, Kibana étant installé sur un Pod d'un cluster Kubernetes (11).

Date Reçue/Date Received 2023-02-06

4. Le procédé selon la revendication 1, dans lequel le premier service (38) comprend Python et la librairie cliente MongoDB dans un Pod scalable sur un cluster Kubernetes (11).

5. Le procédé selon la revendication 1, dans lequel le deuxième service (28) comprend Elasticsearch installé sur un cluster de machine virtuelle ou un cluster Kubernetes (11).

6. Un système de simulation comprenant :

une interface applicative (16) configurée pour recevoir une première demande de simulation (12) d'un utilisateur via un agent (14);

une file d'attente de travail installée dans un Pod d'un cluster Kubernetes (11) pour recevoir la première demande de simulation (12) de l'utilisateur pour répartir une charge de travail sur différentes instances d'un premier service (38); et

un engin analytique installé dans le Pod du cluster Kubernetes (11) pour recevoir des résultats de la simulation et le transmettre (58) à l'interface applicative (16).

7. Le système selon la revendication 6, dans lequel l'interface applicative (16) est un plugin Kibana et utilise Kibana comme serveur applicatif.

8. Le système selon la revendication 6, dans lequel l'engin analytique est configuré pour :

recevoir d'une première demande de simulation (12) d'un agent (14) par une interface applicative (16) et création d'une tâche de simulation (34);

vérifier par le premier service (38) si la demande de simulation (12) est une tâche principale (42),

si oui, le premier service (38) crée des sous tâches (44) par entités pour effectuer la simulation en simultanée et un calcul de la simulation;

sinon, si la tâche (34) est une sous tâche, on calcule (46) la simulation pour une entité correspondante;

Date Reçue/Date Received 2023-02-06

indexer (48) la simulation dans un deuxième service (28);

mettre à jour (50) du statut de la simulation par le premier service (38);

compiler un résultat de génération de langage naturel (52) dans le premier service (38);

interroger (54) un résultat de la simulation de l'agent (14) par l'interface applicative (16) à intervalle régulier jusqu'à ce que la simulation soit complétée;

requérir (56) le résultat de la simulation au deuxième service (28); et

transmettre (58) du résultat de simulation à l'interface applicative (16).

9. Le système selon la revendication 8, dans lequel le premier service (38) comprend Python et MongoDB installés dans des Pods du cluster Kubernetes (11)

10. Le système selon la revendication 8, dans lequel le deuxième service (28) comprend Elasticsearch installé dans des Pods du cluster Kubernetes.

11. Mémoire lisible par ordinateur stockant des instructions exécutables par ordinateur qui, une fois exécutées, exécutent des opérations de :

réception d'une première demande de simulation (12) d'un agent (14) par une interface applicative (16) et création d'une tâche de simulation (34);

vérification par un premier service (38) si la demande de simulation (12) est une tâche principale (42),

si oui, le premier service (38) crée des sous tâches (44) par entités pour effectuer la simulation en simultanée et un calcul de la simulation;

sinon, si la tâche (34) est une sous tâche, on calcule une simulation (46) pour une entité correspondante;

indexation (48) de la simulation dans un deuxième service (28);

Date Reçue/Date Received 2023-02-06

mise à jour (50) du statut de la simulation par le premier service (38);

compilation d'un résultat de génération de langage naturel (52) dans le premier service (38);

interrogation (54) du résultat de la simulation de l'agent (14) par l'interface applicative (16) à intervalle régulier jusqu'à ce que la simulation soit complétée;

requête (56) du résultat de la simulation au deuxième service (28); et

transmission (58) du résultat de simulation à l'interface applicative (16).

Date Reçue/Date Received 2023-02-06

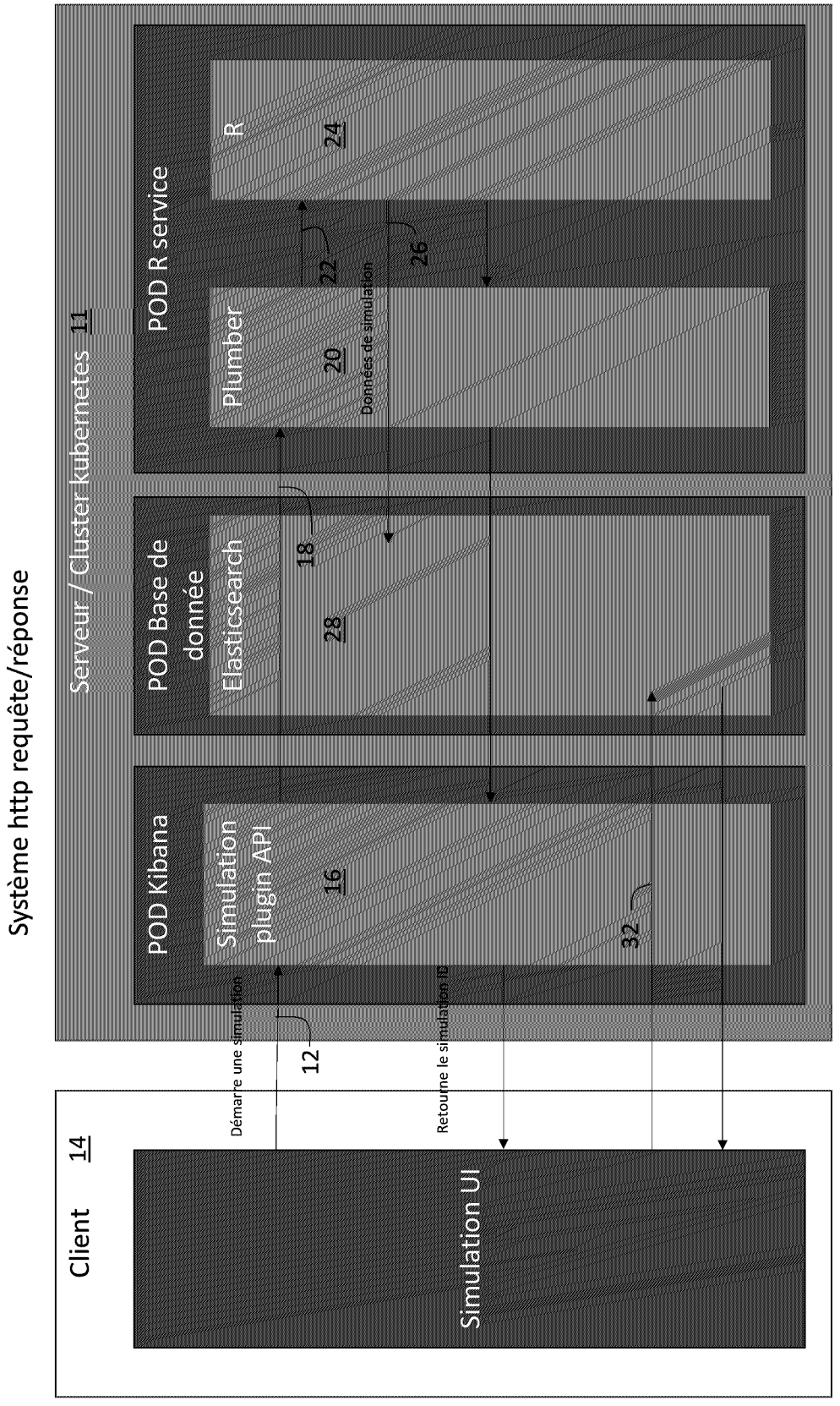

Système http requête/réponse
Serveur / Cluster kubernetes 11
POD Kibana
Simulation plugin API
16
POD Base de donnée
Elasticsearch
18
28
POD R service
Plumber
20
Données de simulation
22
26
R
24
Client 14
Simulation UI
Démarre une simulation
12
Retourne le simulation ID
32

FIG. 1

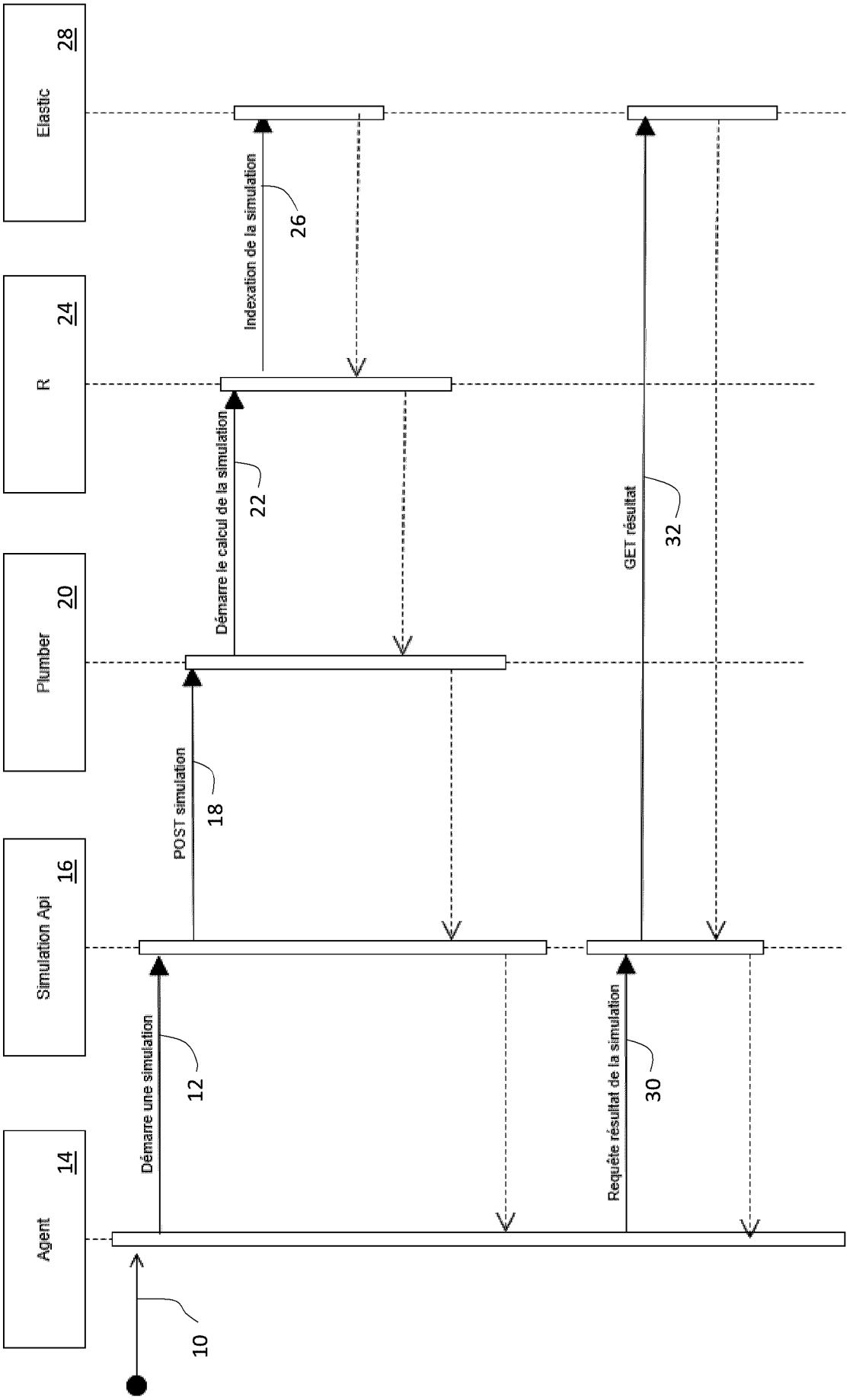


FIG. 2

Date Reçue/Date Received 2023-02-06

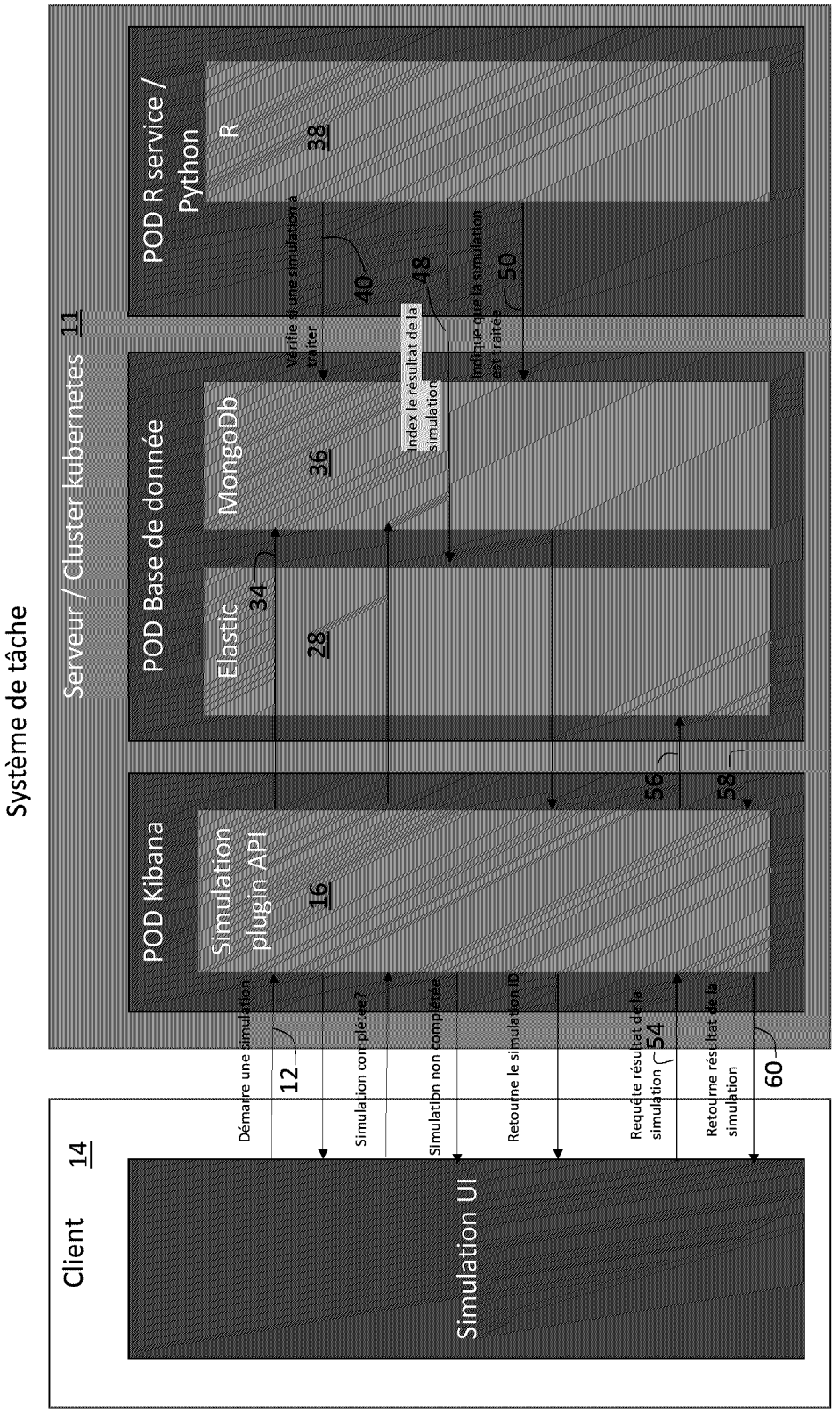

Système de tâche
Serveur / Cluster kubernetes 11
POD Kibana
Simulation plugin API
16
POD Base de donnée
Elastic
34
28
MongoDb
36
POD R service / Python
R
38
Client 14
Simulation UI
Démarre une simulation
12
Vérifie si une simulation à traiter
40
Index le résultat de la simulation
48
Indique que la simulation est traitée
50
Simulation complétée?
Simulation non complétée
Retourne le simulation ID
Requête résultat de la simulation
54
56
Retourne résultat de la simulation
58
60

FIG. 3

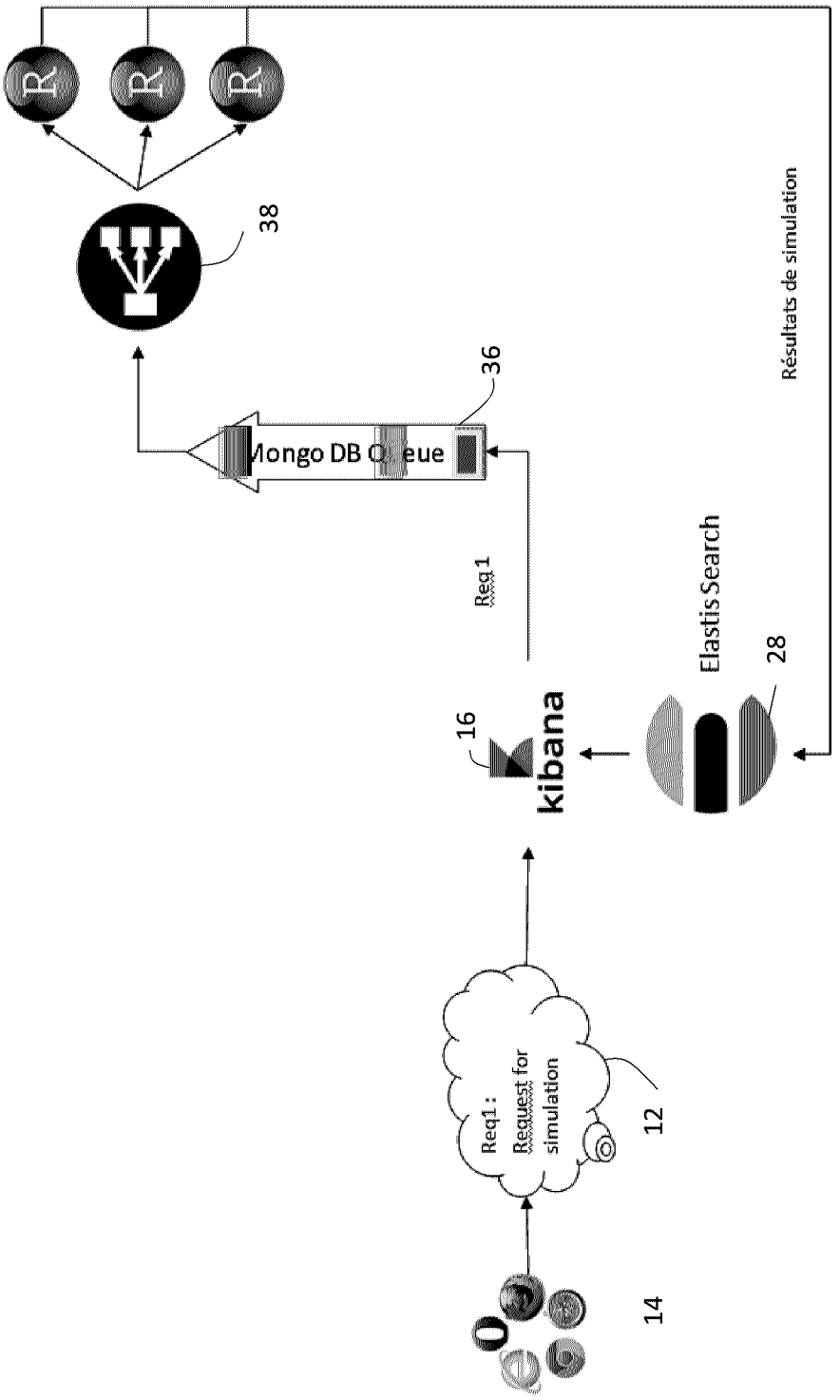

Req1 : Request for simulation
12
14
kibana
16
Req 1
Mongo DB Queue
36
38
Elastis Search
28
Résultats de simulation

FIG. 4

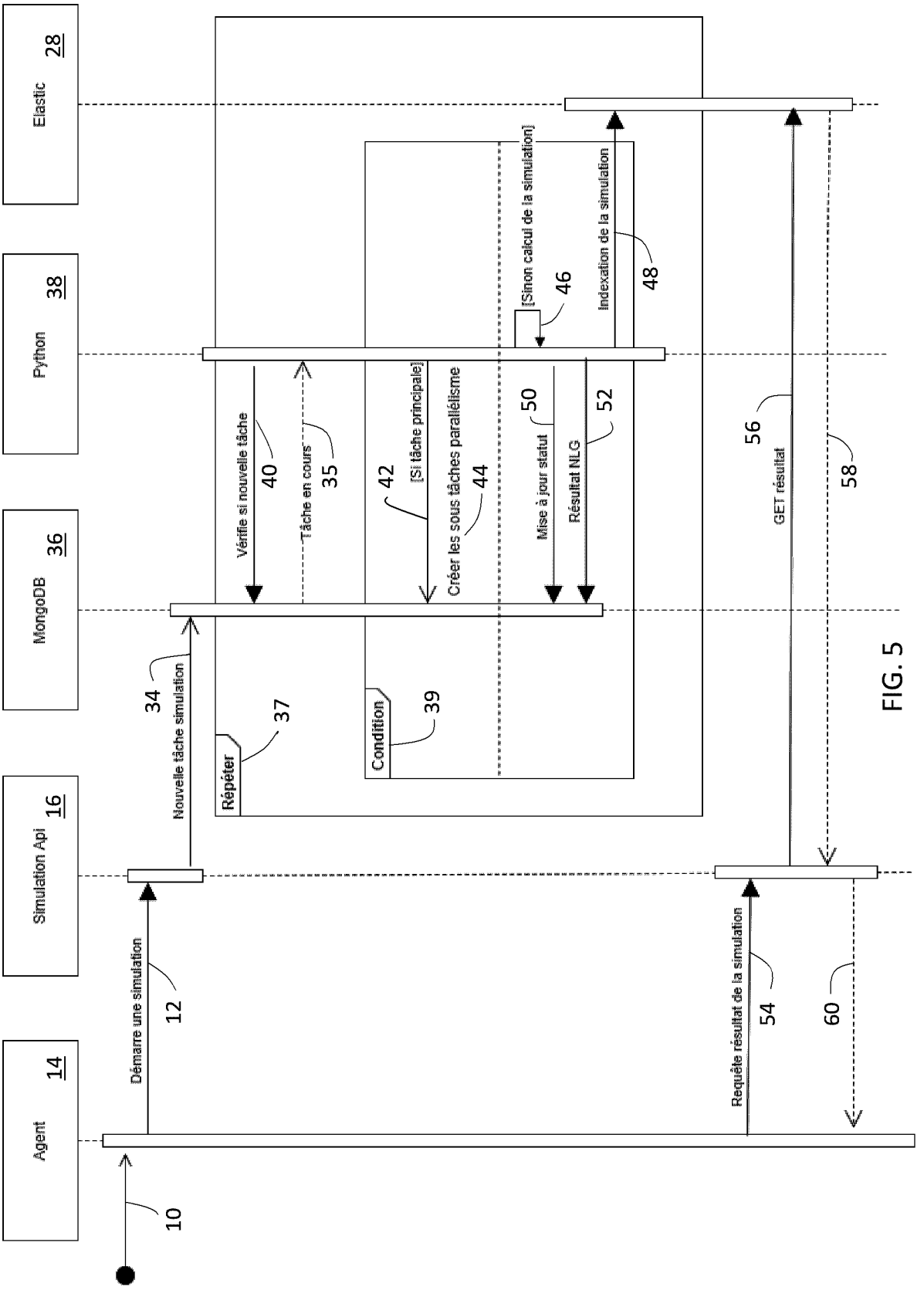

Elastic 28
Python 38
MongoDB 36
Simulation Api 16
Agent 14
10
Démarre une simulation
12
Nouvelle tâche simulation
34
Répéter
37
Condition
39
Vérifie si nouvelle tâche
40
Tâche en cours
35
[Si tâche principale]
42
Créer les sous tâches parallélisme
44
[Sinon calcul de la simulation]
46
Mise à jour statut
50
Résultat NLG
52
Indexation de la simulation
48
GET résultat
56
58
Requête résultat de la simulation
54
60

FIG. 5

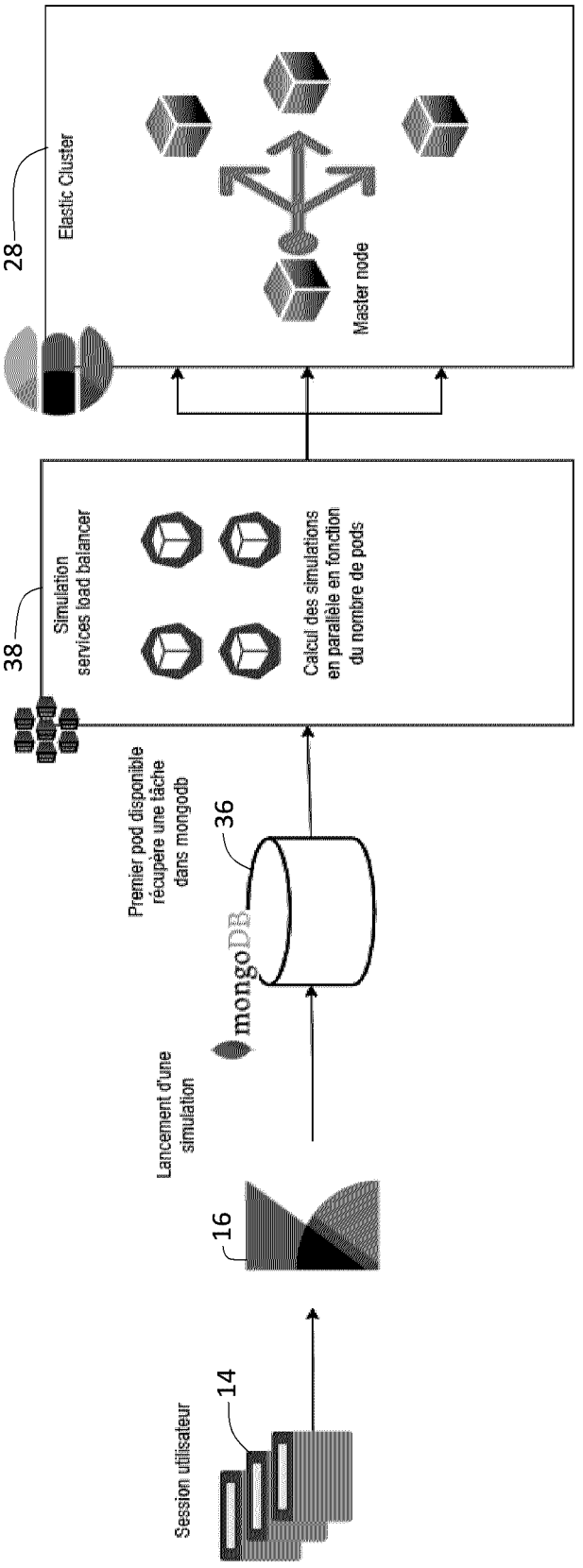
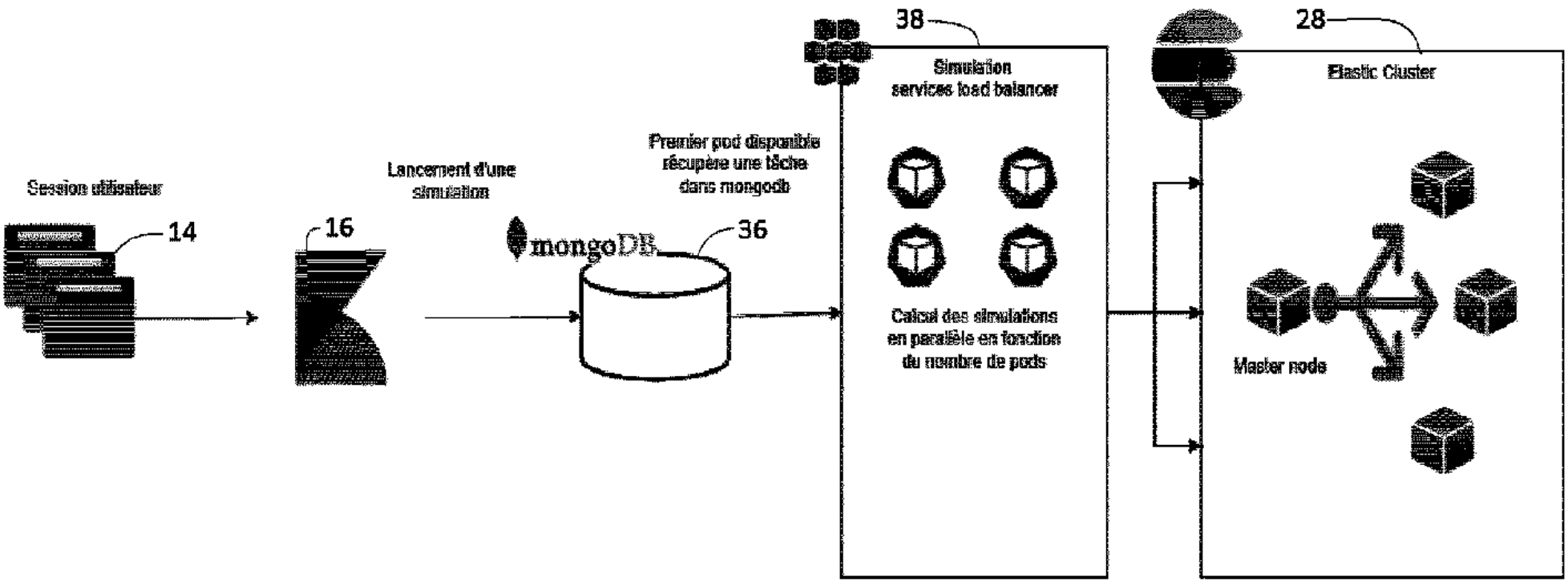

Session utilisateur
14
16
Lancement d'une simulation
mongoDB
36
Premier pod disponible récupère une tâche dans mongodb
38
Simulation services load balancer
Calcul des simulations en parallèle en fonction du nombre de pods
28
Elastic Cluster
Master node

FIG. 6


Session utilisateur
14
16
Lancement d'une simulation
mongoDB
36
Premier pod disponible récupère une tâche dans mongodb
38
Simulation services load balancer
Calcul des simulations en parallèle en fonction du nombre de pods
28
Elastic Cluster
Master node