Raisonnement embarqué et distribué pour le Web des Objets : un état de l'art

Alexandre Bento, Lionel Médini, Kamal Singh, Frederique Laforest

▶ **To cite this version:**

Alexandre Bento, Lionel Médini, Kamal Singh, Frederique Laforest. Raisonnement embarqué et distribué pour le Web des Objets : un état de l'art. Journées Francophones d'Ingénierie des Connaissances (IC) Plate-Forme Intelligence Artificielle (PFIA'21), Jun 2021, Bordeaux, France. pp 48-55. emse-03260470

HAL Id: emse-03260470

https://hal-emse.ccsd.cnrs.fr/emse-03260470

Submitted on 15 Jun 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Raisonnement embarqué et distribué pour le Web des Objets : un état de l'art

Alexandre Bento[1], Lionel Médini[1], Kamal Singh[2], Frédérique Laforest[1]

[1] Université de Lyon, INSA Lyon, UCBL, CNRS, LIRIS UMR 5205, Villeurbanne, France
[2]Université de Lyon, UJM, CNRS, LaHC UMR 5516, Saint-Etienne, France

{alexandre.bento,lionel.medini,frederique.laforest}@liris.cnrs.fr, kamal.singh@univ-st-etienne.fr

Résumé

Le projet Constrained Semantic Web of Things (CoSWoT) a pour objectif de définir une plateforme pour le développement d'applications distribuées et intelligentes pour le Web des Objets utilisant les technologies du Web sémantique. Les graphes de connaissances et le raisonnement à base de règles constituent les éléments clés du projet. Cet article propose un état de l'art des travaux intéressants pour le raisonnement embarqué et distribué dans le cadre du Web des Objets, et trace des lignes directrices pour la mise en place d'un tel raisonnement dans le projet CoSWoT.

Mots-clés

Web des objets, raisonnement, architecture distribuée, optimisation

Abstract

The Constrained Semantic Web of Things (CoSWoT) project aims to define a platform for the developement of smart and distributed applications for the Web of Things that uses the technologies of the semantic Web. Knowledge graphs and rule-based reasoning are the key elements of the project. This paper proposes a state of the art of interesting works on embedded and distributed reasoning in the frame of the Web of Things. It also gives directions to set up such reasoning in the CoSWoT project.

Keywords

Web of things, reasoning, distributed architecture, optimisation

1 Introduction

Le Web des Objets (WoT) [1] désigne les techniques permettant d'utiliser des objets de l'Internet des Objets (IoT) avec les technologies du Web. Différents organismes de standardisation (IETF [2], W3C [3], ETSI [4]) proposent des solutions pour l'interopérabilité sémantique dans l'IoT.
Les objectifs du projet CoSWoT [5] sont de proposer une architecture logicielle distribuée compatible avec le WoT et embarquée dans des dispositifs contraints en ressources (capacité de calcul, mémoire, énergie...). Cette architecture ajoutera des fonctionnalités de raisonnement à des dispositifs de capacités diverses, en répartissant les tâches de traitement entre les dispositifs.
L'analyse des cas d'usage du projet dans les domaines du batiment intelligent et de l'e-agriculture ont permis d'identifier que la plateforme doit fournir un raisonnement à base de règles sur des données dynamiques représentées sous la forme de multiples graphes de connaissances. Plus précisément, les besoins métier sont : (i) un raisonnement incrémental et distribué sur des règles métier incluant la détection de dépassements de seuils issus de l'agrégation temporelle et spatiale de données, (ii) l'appel à des fonctions externes (par exemple, des calculs arithmétiques) dans la tête des règles, (iii) le traitement de flux de données, et (iv) l'intégration du processus de raisonnement dans d'autres outils de la plateforme CoSWoT (stockage, traçabilité...).
Le raisonnement distribué a fait l'objet d'une attention particulière au cours des dernières décennies [1, 2] pour traiter de très grands ensembles de données RDF sur des machines de même architecture dans des clusters ou sur Internet [3, 4]. La distribution dans les architectures edge [6] ou fog [7] porte sur des configurations très différentes [5]. Tout d'abord, les dispositifs impliqués ont des capacités de calcul hétérogènes et des architectures matérielles variées. Deuxièmement, les dispositifs ont une connectivité plus ou moins limitée. Ces caractéristiques invalident les résultats de recherches antérieures, et soulèvent le défi de développer des stratégies de raisonnement efficaces en termes de distribution des données et des tâches de raisonnement dans un contexte d'hétérogénéité, et aussi près des sources de données que possible.
Dans cet article, nous présentons l'état de l'art de la recherche en raisonnement dans les domaines qui concernent les contraintes du projet CoSWoT : objets contraints, raisonnement local, et raisonnement distribué. Nous concluons ensuite avec quelques pistes de recherche.
Pour des raisons de place, nous partons du principe que le lecteur connait le domaine du Web sémantique [6] et les

1. https://www.w3.org/WoT/
2. https://www.ietf.org/
3. https://www.w3.org/
4. https://www.etsi.org/
5. https://coswot.gitlab.io/
6. https://fr.wikipedia.org/wiki/Edge_computing
7. https://fr.wikipedia.org/wiki/Fog_computing

principes du raisonnement à base de règles [7].

2 Raisonnement et objets contraints

2.1 Contraintes liées à l'IoT

Les objets contraints ou autonomes ont différentes caractéristiques qui rendent difficiles la communication et l'exécution de tâches complexes [8, 9]. Pour instancier un raisonneur, choisir les règles à y déployer et exécuter un algorithme de raisonnement sur de tels objets, il faut prendre en compte les différentes dimensions identifiées ci-dessous.

Architecture matérielle. Certains dispositifs disposent d'unités de traitement aux architectures non standards ou aux capacités de calcul limitées. [10] cite le cas de dispositifs de type microcontrôleurs au bus de données de 8 bits. De telles unités peuvent ralentir, voire empêcher le traitement de règles "à forte expressivité", c'est-à-dire plus susceptibles de produire des faits qui eux-mêmes déclencheront l'exécution de règles. Il existe donc un lien entre la complexité maximale atteignable par un algorithme de raisonnement et le type de processeur ou microcontrôleur sur lequel ce raisonnement peut être déployé. Cela peut se produire en particulier lorsque les données arrivent en flux et qu'il faut les traiter à la volée.
Par ailleurs, comme tous les algorithmes, la chaîne de déploiement doit être adaptée au portage du code source du raisonneur sur l'architecture matérielle de chaque dispositif cible. La diversité des dispositifs utilisés dans une architecture IoT complexifie cette chaîne. Enfin, il faut bien entendu que le code puisse être déployé sur le support de persistance du dispositif (ROM, EEPROM, Flash), c'est-à-dire que sa taille soit inférieure à celle de ce support [8].

Mémoire de travail. Certains dispositifs ne peuvent ni stocker ni travailler sur des graphes de connaissances volumineux. [8] propose également une catégorisation des objets en fonction de la taille de la mémoire de travail disponible [8]. Cette contrainte s'applique à la fois à la taille du graphe d'entrée (faits explicites) mais aussi au graphe déduit (faits explicites + implicites). Certains cas d'utilisation imposent de raisonner sur des flux de données correspondant par exemple à une fenêtre temporelle ; la taille de celle-ci peut aussi être limitée par la mémoire disponible.

Énergie. Le raisonnement sur des dispositifs autonomes en énergie peut être limité par deux aspects : la communication (en particulier sans fil), qui représente la principale source de consommation énergétique d'un dispositif, ainsi que l'intensivité des calculs, dont le coût est moindre et qui peut s'avérer rentable si elle permet de diminuer la charge utile des communications réseau [11]. Dans la perspective d'un algorithme de raisonnement distribué, il convient donc de minimiser les communications entre les dispositifs en évaluant le coût des différentes tâches de raisonnement et leur faisabilité en local.

Connectivité. De nombreux travaux ont été menés autour des problématiques de transmission de données entre nœuds d'un réseau dans l'IoT [8, 9, 10]. Un nœud à la connectivité limitée, notamment pour des raisons d'économie d'énergie, ne prendra probablement pas part à un protocole de raisonnement distribué pour traiter des données provenant d'autres sources. Mais pour les raisons exposées plus haut, il peut aussi avoir des difficultés à raisonner sur ses propres données. C'est pourquoi, pour certains capteurs isolés et connectés périodiquement, il faut savoir choisir entre délivrer des données brutes, des observations sémantisées ou des faits implicites issus de déductions à partir de ces observations.

2.2 Déploiement sur des objets contraints

Les technologies du Web sémantique, et en particulier les raisonneurs classiques (voir section 3) sont trop gourmands en ressources pour être portés directement sur des dispositifs contraints. Il n'existe par exemple que quelques travaux qui intègrent du raisonnement dans des dispositifs contraints, et plusieurs d'entre eux sont conçus pour les téléphones mobiles [12, 13, 14, 15] et non pour des dispositifs plus contraints comme des capteurs [16, 17] ou des microcontrôleurs. Dans cette partie, nous présentons les travaux de la littérature qui implémentent les différentes parties de la "stack" Web sémantique et raisonnement sur des objets contraints.

Échange de données RDF. Plusieurs implémentations proposent la gestion de données RDF dans des objets contraints. Wiselib [16] permet à un microcontrôleur de type iSense [9] d'héberger et de servir directement à ses clients des triplets RDF à l'aide des protocoles CoAP et 6LowPan. [18] propose un serveur RDF utilisant CoAP et JSON-LD pour interagir avec les capteurs et les actionneurs déployés sur un microcontrôleur encore plus contraint : Arduino Uno [10]. Plus généralement, [19] propose un format de sérialisation de données RDF en binaire qui s'appuie sur EXI4JSON et CBOR pour réduire considérablement la taille des faits échangés entre un raisonneur et un client ou entre plusieurs raisonneurs.

Outils de traitement pour RDF. Pour pouvoir déployer un moteur d'inférence sur un objet, il faut que les outils sous-jacents soient disponibles dans l'environnement d'exécution de l'objet. C'est pourquoi nous recensons les outils de stockage et traitement de graphes RDF en fonction des langages de programmation dans lesquels ils sont implémentés et de leur disponibilité sur les différentes plateformes matérielles.

Les outils de base des technologies du Linked Data et du Web sémantique ont des implémentations bien connues telles que Jena [11], librdf [12], N3.js [13] ou RDFlib [14]. En termes de performances, [20] a démontré que la bibliothèque RDF

8. Cette classification date de 2014, les seuils définis ne sont plus à jour.

9. https://www.quarbz.com/Wireless%20Sensor%20Network/2.%20iSense%20Devices%20and%20Modules.pdf
10. https://store.arduino.cc/arduino-uno-rev3
11. Implémenté en Java, http://jena.apache.org/
12. Implémenté en C, http://librdf.org/
13. Implémenté en JavaScript, https://github.com/rdfjs/N3.js
14. Implémenté en Python, https://rdflib.readthedocs.io

Sophia [15] développée en Rust [16], est plus rapide pour charger un graphe RDF en mémoire et répondre à des requêtes simples sur ce graphe. À l'heure de la rédaction de ce document, les chaînes de compilation et de déploiement de code Rust sur les plateformes matérielles d'objets contraints sont de plus en plus répandues, stables et disponibles pour une part de plus en plus importante de ces plateformes.

Dans le même esprit, WASMTree [21] est une implémentation en Rust et Web Assembly qui permet de construire des triplestores RDF en JavaScript. Elle permet de stocker et interroger des datasets RDF en utilisant une stratégie d'indexation intelligente, et surpasse les bibliothèques de référence. Cependant, elle est limitée au stockage et à la récupération de quads RDF au niveau syntaxique, et des travaux supplémentaires sont nécessaires pour travailler au niveau sémantique requis par le raisonnement à base de règles.

Raisonnement sur mobile et Web. Même si les smartphones ont des capacités de calcul beaucoup plus élevées que les appareils généralement considérés comme contraints, nous mentionnons ici le déploiement de tâches de raisonnement sur des appareils mobiles. [12] et [15] ont adapté des raisonneurs de référence sur des téléphones Android, et ont montré que ces machines n'avaient pas les capacités de calcul nécessaires pour exécuter correctement des tâches de raisonnement. Bien que les spécifications des appareils mobiles aient évolué depuis [17], ces travaux montrent que les raisonneurs existants ne sont pas adaptés aux dispositifs contraints. [13] présente un raisonneur développé sur iOS pour iPhone qui offre des performances décentes, mais la principale optimisation présentée provient de l'utilisation du langage de programmation Swift natif d'iOS, au lieu de langages standards. La généricité de cette approche est donc discutable. Dans le même ordre d'idée, le raisonneur HyLAR [22] décrit dans les sections suivantes peut s'exécuter dans un client Web, éventuellement mobile. Programmé en JavaScript pour être portable, il peut s'exécuter indifféremment côté serveur et côté client, et a été porté dans un Web worker [18] pour améliorer ses performances. Mais il nécessite une machine virtuelle JS pour s'exécuter, qui n'est en général pas disponible pour les objets contraints.

Raisonnement et architectures matérielles. Sans avoir été spécifiquement développé pour les objets contraints, Inferray [23] accélère également le raisonnement en l'optimisant en fonction de la taille des mots-mémoire de l'achitecture sur laquelle il est déployé. Il utilise une structure de données basée sur des tableaux pour l'inférence in-memory en utilisant des algorithmes de tri-jointure et de tri-comptage personnalisés. Il gère l'inférence des ensembles de règles RDFS, ρdf, et RDFS-Plus et a montré de très bonnes performances sur certains ensembles de données. Cependant, il ne supporte pas la suppression des faits.

15. https://github.com/pchampin/sophia_rs
16. https://www.rust-lang.org/
17. Nous n'avons pas trouvé de travaux plus récents sur cette question dans la litérature.
18. https://html.spec.whatwg.org/multipage/workers.html#workers

3 Raisonnement local

En termes de performances, l'algorithme de raisonnement naïf bouclant sur les règles est loin d'être optimal lorsque les règles sont interdépendantes. Plusieurs optimisations sont présentées ci-dessous.

3.1 Optimisations du raisonnement local

RETE [24] structure les règles sous la forme d'un arbre de préfixes nommé "trie" [19]. Le trie RETE est composé de deux types de nœuds [20] (Figure 1). Les nœuds alpha représentent les conditions atomiques dans le corps des règles. Les nœuds bêta effectuent des opérations de jointure entre les sorties de deux autres nœuds. Les faits compatibles activent les nœuds du trie et le résultat est conservé en mémoire. Ainsi, lorsque de nouveaux faits arrivent, il n'est pas nécessaire de réévaluer les faits précédents. Le raisonnement est finalisé quand les nœuds feuilles sont atteints, c'est-à-dire quand tous les nœuds alpha d'une règle ont été activés et toutes les opérations de jointure des nœuds bêta ont été effectuées. L'algorithme RETE n'explore qu'une partie de l'arbre de règles, au lieu de parcourir chaque règle pour chaque fait explicite ; cela permet d'augmenter la vitesse d'exécution, au prix de l'empreinte mémoire. Diverses optimisations ont été proposées pour l'algorithme RETE, dont certaines sont présentées ci-après.

$\text{RETE}_{\text{pool}}$ [14] réduit l'empreinte mémoire de RETE en limitant la duplication des données pendant le raisonnement. Il utilise une mémoire partagée pour tous les nœuds alpha du réseau. De cette façon, les doublons sont éliminés à l'insertion. Dans les cas où un triplestore RDF est utilisé avec le raisonneur, un autre niveau de duplication est éliminé, chaque nœud alpha ayant des références aux triplets contenus dans le triplestore plutôt qu'une copie locale. Cela permet d'économiser de la mémoire, au détriment de la vitesse d'exécution.

COROR [17] réduit la consommation mémoire de RETE. Il crée un graphe de dépendances entre les règles, puis charge en mémoire uniquement les règles nécessaires : pour une ontologie et un jeu de règles donnés, les règles ne pouvant pas aboutir à la création de nouveaux faits ne sont pas considérées. Ensuite, il décompose l'algorithme RETE en deux phases. La première phase effectue un appariement entre les faits et les conditions des nœuds alpha, comme le fait l'algorithme RETE classique. La seconde restructure l'arbre RETE en utilisant les statistiques de la première phase : les règles et les conditions sont réorganisées pour que les plus sélectives soient analysées en premier. COROR permet de réduire l'empreinte mémoire de 74 % en moyenne, par rapport à l'implémentation de référence.

3.2 Raisonnement incrémental

Au fil du temps, les données d'une application peuvent évoluer, modifiant le graphe de connaissances sur lequel s'effectue le raisonnement. Les faits implicites qui en découlent sont donc impactés par ces modifications. Parmi

19. https://en.wikipedia.org/wiki/Trie
20. https://www.sparklinglogic.com/rete-algorithm-demystified-part-2/

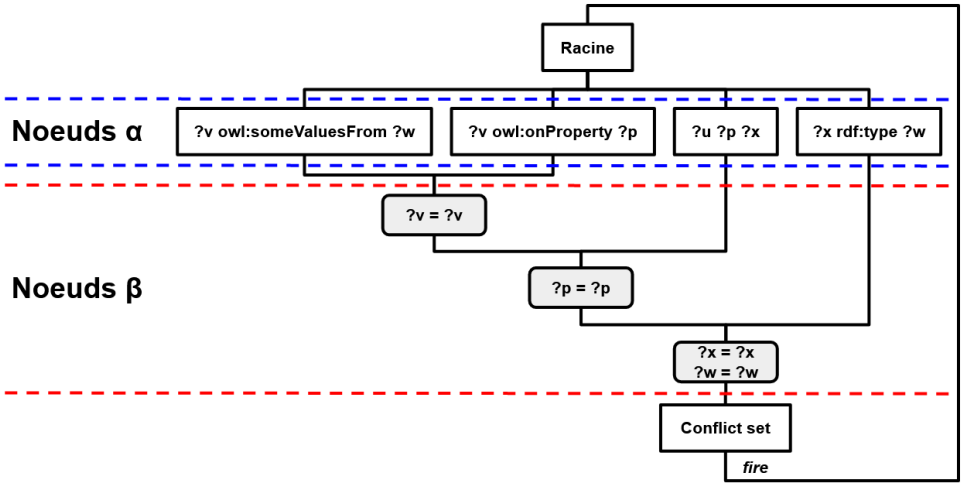

FIGURE 1 – Un exemple d'arbre RETE pour la règle
(?v owl:someValuesFrom ?w) ∧ (?v owl:onProperty ?p) ∧ (?u ?p ?x) ∧ (?x rdf:type ?w) → (?u rdf:type ?v).

les algorithmes de matérialisation présentés ci-avant, tous ne supportent pas l'insertion de faits, et aucun d'entre eux n'en supporte la suppression. Pour résoudre ce problème, des raisonneurs incrémentaux ont été définis ; ils permettent à la fois l'insertion et la suppression (aussi appelée maintenance incrémentale) des faits explicites.

Lorsqu'un fait explicite est supprimé, l'algorithme Delete / Rederive (DRed) [25] supprime d'abord tous les faits implicites qui en dépendent. Ensuite, il relance le raisonnement sur les faits non supprimés, ce qui permet de redériver certains faits supprimés. D'autres travaux utilisent une variante de l'algorithme DRed, comme [26].

Pour permettre le raisonnement incrémental, RDFox [27] utilise l'algorithme Backward/Forward : lorsqu'un fait explicite est supprimé, il cherche immédiatement des dérivations alternatives pour les faits qui en découlent. En comparaison avec DRed, le gain en performance est particulièrement visible avec les faits implicites qui découlent de nombreuses déductions en chaîne (par exemple, rdfs:subClassOf). GraphDB [21] utilise la même approche. La Figure 2 illustre les approches DRed et Backward/Forward. Dans cet exemple, lorsque le fait E_1 est supprimé, DRed supprime I_1 et I_2 pour les redériver ensuite via E_2, tandis que Backward/Forward identifie la dérivation de I_1 via E_2 et ne le supprime donc pas.

HyLAR+ [28] propose une approche de raisonnement incrémental dite "à base de tags". Lorsqu'un fait explicite est retiré, il n'est pas supprimé mais marqué comme invalide ; aucun autre calcul n'est nécessaire pour traiter la suppression des faits implicites. Lors d'une requête, les faits explicites invalides et les faits implicites dont les conditions ne sont plus remplies sont filtrés. Lors de multiples cycles d'insertions / suppressions, HyLAR+ permet un gain en temps jusqu'à plus de 80%. Cette approche est efficace lorsque des faits apparaissent et disparaissent régulièrement, comme dans le cas de dépassements de seuils. Elle est moins adaptée pour des mesures brutes. Comme RETE, elle a tendance à privilégier la performance au détriment de l'espace mémoire.

21. https://graphdb.ontotext.com/documentation/free/reasoning.html#retraction-of-assertions

3.3 Raisonnement sur flux

Dans le cas d'applications utilisant des données arrivant en flux, celles-ci doivent être traitées en temps réel ou quasi-réel. Des compromis doivent être trouvés entre la complexité du raisonnement demandé et la vitesse de traitement des données. Les ingrédients du raisonnement sur flux proviennent des systèmes de gestion des flux de données et des systèmes de traitement des événements complexes [29]. La plupart des approches utilisent un opérateur de fenêtrage temporel pour effectuer le raisonnement sur des sous-ensembles de données.

Slider [30] est un raisonneur sur flux à chaînage avant multithread optimisé en mémoire. Chaque thread correspond à une règle et une règle peut instancier plusieurs threads. Chaque règle a une mémoire tampon dédiée qui stocke les faits du flux entrant avant que chacun soit traité par un de ses threads ; les faits implicites et explicites sont stockés dans un triplestore partagé par tous les threads, et avec une gestion fine de la concurrence d'accès. Cependant, Slider ne gère pas de fenêtres temporelles et ne permet que l'insertion de faits.

IMaRS (Incremental Materialization for RDF Streams) [31] suppose que les flux RDF sont constitués de triplets horodatés. Comme il utilise une fenêtre glissante fixe, il associe un cachet d'expiration à chaque fait. IMaRS est d'un ordre de grandeur plus rapide que DRed jusqu'à 0,1 % de changements dans les données, et deux ordres plus rapides jusqu'à 2,5 %. Cependant, dans le cas où les données changent de plus de 13 %, il obtient des performances pires qu'une approche naïve où la matérialisation est effectuée à chaque fois que le contenu de la fenêtre change.

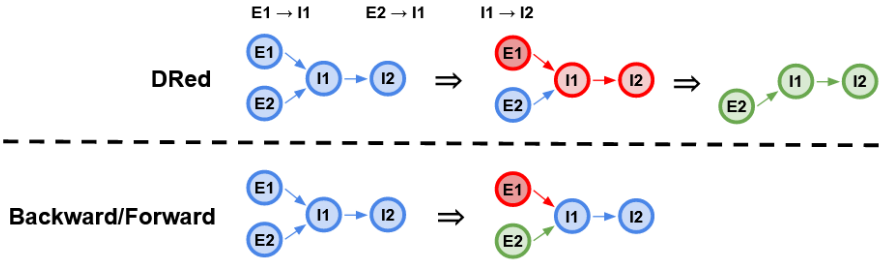

FIGURE 2 – Un exemple de raisonnement incrémental avec DRed et Backward/Forward. Les suppressions sont présentées en rouge, et les redérivations en vert.

4 Raisonnement distribué

4.1 Parallélisation

RDFox [27] est un algorithme rapide de raisonnement parallélisé qui distribue uniformément la charge de travail sur plusieurs cœurs d'un processeur. Plus précisément, les faits explicites et implicites sont ordonnés dans l'ABox. Un cœur extrait de l'ABox un fait qui n'a pas encore été traité. S'il satisfait une condition d'une règle, RDFox essaie d'appliquer les autres conditions sur les faits qui le précèdent dans l'ABox. Il utilise pour l'ABox un schéma d'indexation rapide basé sur des tables de hachage : les triplets RDF sont encodés dans un tableau à six colonnes, contenant les trois ressources du triplet et trois pointeurs vers des listes chaînées de triplets contenant respectivement le même sujet, prédicat ou objet. Cette structure permet un accès en lecture très rapide. Cependant, tous les cœurs partagent le même espace mémoire, ce qui n'est pas transposable à une architecture distribuée. Par ailleurs, l'optimisation de l'inférence de RDFox ne concerne pas l'ordre des faits et des règles, mais implémente un réordonnancement glouton des conditions atomiques dans le corps de chaque règle.

[32, 33] proposent des méthodes pour le traitement parallèle de règles par partitionnement et distribution de la charge de travail. Leur méthode est basée sur RETE et consiste à exécuter indépendamment et simultanément les nœuds alpha et bêta de l'algorithme RETE. Par exemple, les jointures des nœuds bêta d'une même profondeur peuvent être exécutées indépendamment et donc simultanément.

4.2 Distribution

Distribution des données. Lorsque les données sont distribuées, les différents raisonneurs interagissent les uns avec les autres en échangeant des messages contenant des faits implicites et explicites. Il faut veiller à ce que le coût d'échange des messages ne soit pas supérieur au coût d'envoi des données à un emplacement central, ou que ce coût soit contre-balancé par d'autres avantages.

Le sharding [34] fait référence au partitionnement horizontal des données. Lorsque les données sont partitionnées de cette manière, chaque partition de données est traitée ou exploitée séparément. De cette façon, il est possible de répartir la charge sur différentes machines et d'augmenter la fiabilité en évitant les points uniques de défaillance. Cette technique est valable dans les cas où il est possible d'identifier des partitions indépendantes.

Le raisonnement contextuel [35] permet de raisonner avec différents points de vue comme dans le cas d'une fédération de données, ou de raisonner avec différentes croyances (systèmes multi-agents). On considère ici des ontologies distribuées qui décrivent le monde selon différents points de vue ; chaque ontologie décrit le contexte du nœud où elle se trouve. Ces ontologies peuvent être construites et évoluer indépendamment les unes des autres. Il est alors parfois nécessaire d'établir des alignements entre les différentes ontologies locales. [36] propose un algorithme pour effectuer un raisonnement contextuel sur un réseau d'ontologies alignées. Leur architecture est distribuée et un raisonneur global traite les alignements. Le raisonneur global communique avec les autres raisonneurs en utilisant OWL-Link [37]. Chaque nœud utilise HermiT [38] comme raisonneur OWL.

Distribution des règles. EDR [39] est une approche de raisonnement distribué basée sur le paradigme REST. Sa topologie est hiérarchique. Les nœuds déclarent les données qu'ils peuvent fournir ainsi que les données qu'ils requièrent. EDR décompose les règles en différentes parties. Chaque nœud utilise un moteur d'inférences basé sur SHACL, qui lui permet soit d'appliquer lui-même la règle reçue, soit de transférer la règle à un nœud fils. Au final, les règles sont installées aussi profondément que possible dans la topologie du réseau. On notera que les nœuds avec EDR n'ont connaissance que de leurs voisins immédiats. De plus, EDR ne prend pas en compte les changements de topologie ni les défaillances de nœuds. EDR est donc difficilement adaptable. Le placement des règles ne tient pas compte non plus des contraintes de mémoire et d'énergie.

Distribution des tâches de raisonnement. MORe [40] est un méta-raisonneur qui répartit les tâches de classification d'ontologies entre un raisonneur OWL2-EL et un raisonneur OWL2-Full. Ce dernier étant beaucoup plus lourd, MORe réserve son usage aux parties qui ne peuvent pas être traitées par OWL2-EL. Le raisonnement est ainsi plus efficace. Le résultat est l'union logique des résultats des deux raisonneurs.

HyLAR-Framework [22] [22] permet aux applications Web

22. https://github.com/ucbl/HyLAR-Framework/

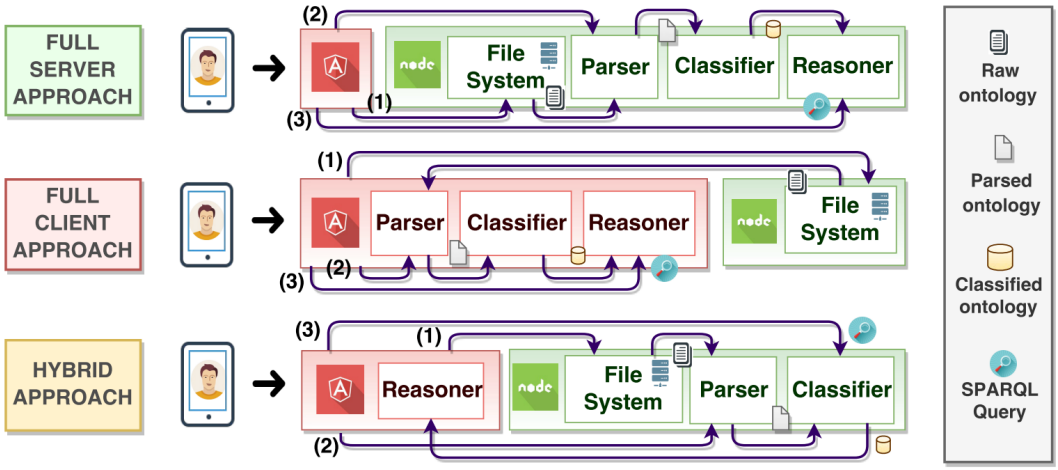

FIGURE 3 – Répartitions possibles des tâches de raisonnement avec le framework HyLAR [22]

d'effectuer des tâches de raisonnement indifféremment côté serveur ou côté client, en déployant le même raisonneur [23] aux deux endroits. Une utilisation classique est de réaliser les opérations lourdes et identiques pour tous les clients côté serveur, puis de déléguer le reste des inférences aux clients, en fonction des mises à jour incrémentales de leurs bases de connaissances locales (Figure 3). Ce framework peut aussi déterminer automatiquement s'il sera plus efficace de traiter les règles côté serveur et d'envoyer ensuite la fermeture déductive du graphe au client, ou de n'envoyer que sa réduction et de déléguer le raisonnement au client. Pour cela, il évalue les capacités de calcul du client et la qualité de sa connexion réseau, en lançant un "benchmark" correspondant à un calcul simple. Les information obtenues à l'aide de ce benchmark sont toutefois d'une qualité assez basique, compte tenu des mécanismes de protection des données en vigueur dans les API JavaScript côté client.

5 Conclusion et questions ouvertes

Le raisonnement réparti dans les architectures fog ou edge pour le Web des Objets pose de nouveaux défis, qui n'ont pour l'instant pas trouvé de solution dans la littérature. Dans cet article, nous avons identifié des problématiques et les travaux qui peuvent aider à résoudre ces défis, sur les thématiques du raisonnement embarqué, de l'optimisation du raisonnement local, du raisonnement incrémental, de la gestion de flux de données, et du raisonnement distribué. Mais de nombreux défis restent à relever, nous en citons quelques uns ci-après.

Il existe quelques approches de raisonnement embarqué dans des dispositifs contraints. Un des défis est de concevoir un raisonneur efficace en termes de mémoire, de bande passante et de consommation d'énergie. Il doit avoir une faible empreinte et définir des structures de données et algorithmes de raisonnement les plus efficaces. Celle-ci devra également être efficace en termes de consommation d'énergie et éviter une trop grande quantité de données échangées.

23. HyLAR [22], implémenté en JavaScript.

Une façon de réduire les échanges de données pourrait également consister à échanger des connaissances dans de nouveaux formats compressés. Une majorité des contributions étudiées ont été évaluées sur des smartphones, dont la puissance de calcul est supérieure à ce que nous considérons comme des appareils contraints. Ainsi, les raisonneurs développés dans ce contexte ne correspondent pas à notre projet dédié au Web des Objets. Néanmoins, les smartphones fonctionnent sur batterie et l'énergie reste une ressource critique à optimiser. Ainsi, certaines propositions faites pour les smartphones pourront être reprises pour points de départ. Par ailleurs, dans le contexte du Web des Objets et du projet CoSWoT en particulier, il est nécessaire de concevoir des raisonneurs distribués, capables de travailler sur des données distribuées.

Au sujet du raisonnement distribué, un partitionnement devra être défini pour répartir le travail entre les nœuds. Ce partitionnement pourra concerner tant les données que les règles. Plusieurs options seront étudiées. Par exemple, on peut définir des graphes de connaissances correspondant chacun à une "couche d'information" (matériel, plateforme, domaine d'application, configuration, préférences de l'utilisateur, etc.).

Une autre question ouverte est liée à l'agrégation des données. Certains cas d'utilisation de CoSWoT nécessitent de calculer des valeurs moyennes sur les données en continu, dans une fenêtre temporelle et géographique donnée. Comment ces fonctions d'agrégation seront-elles intégrées au processus de raisonnement ? Si l'agrégation est séparée du raisonnement, comment les deux processus seront-ils articulés ?

La littérature scientifique doit également encore être explorée dans d'autres domaines, notamment l'indexation efficace de triplets, l'optimisation de l'ordonnancement des règles, l'encodage des données en mémoire, l'optimisation logicielle orientée matériel ou encore l'optimisation multi-objectifs.

Remerciements

Ce travail est soutenu par la subvention ANR-19-CE23-0012 de l'Agence Nationale de la Recherche pour le projet CoSWoT.

Références

[1] Philippe Adjiman, Philippe Chatalic, François Goasdoué, Marie-Christine Rousset, and Laurent Simon. Distributed reasoning in a peer-to-peer setting: Application to the semantic web. *Journal of Artificial Intelligence Research*, 25:269–314, 2006.

[2] L Serafini and A Tamilin. Distributed reasoning architecture for the semantic web. In *Proceedings of the Second European Semantic Web Conference, ESWC*, 2005.

[3] Eyal Oren, Spyros Kotoulas, George Anadiotis, Ronny Siebes, Annette ten Teije, and Frank van Harmelen. Marvin: Distributed reasoning over large-scale semantic web data. *Journal of Web Semantics*, 7(4):305–316, 2009.

[4] Aidan Hogan, Jeff Z Pan, Axel Polleres, and Stefan Decker. Saor: template rule optimisations for distributed reasoning over 1 billion linked data triples. In *International Semantic Web Conference*, pages 337–353. Springer, 2010.

[5] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98:289–330, 2019.

[6] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific american*, 284(5):34–43, 2001.

[7] Pedro Barahona, François Bry, Enrico Franconi, Nicola Henze, and Ulrike Sattler. *Reasoning Web: Second International Summer School 2006, Lisbon, Portugal, September 4-8, 2006, Tutorial Lectures*, volume 4126. Springer, 2006.

[8] Carsten Bormann, Mehmet Ersue, and Ari Keranen. Terminology for constrained-node networks. *Internet Engineering Task Force (IETF): Fremont, CA, USA*, pages 2070–1721, 2014.

[9] Asma Haroon, Munam Ali Shah, Yousra Asim, Wajeeha Naeem, Muhammad Kamran, and Qaisar Javaid. Constraints in the iot: the world in 2020 and beyond. *Constraints*, 7(11):252–271, 2016.

[10] Zach Shelby, Klaus Hartke, and Carsten Bormann. The constrained application protocol (coap). 2014. *URL https://tools. ietf. org/html/rfc7252*, 2014.

[11] Christopher M Sadler and Margaret Martonosi. Data compression algorithms for energy-constrained devices in delay tolerant networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 265–278, 2006.

[12] Carlos Bobed, Roberto Yus, Fernando Bobillo, and Eduardo Mena. Semantic reasoning on mobile devices: Do androids dream of efficient reasoners ? *Journal of Web Semantics*, 35:167–183, 2015.

[13] Michele Ruta, Floriano Scioscia, Filippo Gramegna, Ivano Bilenchi, and Eugenio Di Sciascio. Mini-me swift: the first mobile owl reasoner for ios. In *European Semantic Web Conference*, pages 298–313. Springer, 2019.

[14] William Van Woensel and Syed Sibte Raza Abidi. Optimizing semantic reasoning on memory-constrained platforms using the rete algorithm. In *European Semantic Web Conference*, pages 682–696. Springer, 2018.

[15] Roberto Yus, Carlos Bobed, Guillermo Esteban, Fernando Bobillo, and Eduardo Mena. Android goes semantic: Dl reasoners on smartphones. In *Ore*, pages 46–52, 2013.

[16] Henning Hasemann, Alexander Kröller, and Max Pagel. Rdf provisioning for the internet of things. In *2012 3rd IEEE International Conference on the Internet of Things*, pages 143–150. IEEE, 2012.

[17] Wei Tai, John Keeney, and Declan O'Sullivan. Resource-constrained reasoning using a reasoner composition approach. *Semantic Web*, 6(1):35–59, 2015.

[18] Remy Rojas, Lionel Médini, and Amélie Cordier. Toward Constrained Semantic WoT. In *Seventh International Workshop on the Web of Things (WoT 2016)*, pages 31 – 37, Stuttgart, Germany, November 2016. W3C, ACM New York, NY, USA.

[19] Victor Charpenay, Sebastian Käbisch, and Harald Kosch. Towards a binary object notation for rdf. In *European Semantic Web Conference*, pages 97–111. Springer, 2018.

[20] Pierre-Antoine Champin. Sophia: a Linked Data and Semantic Web toolkit for Rust. In Erik Wilde and Mike Amundsen, editors, *The Web Conference 2020: Developers Track*, Taipei, Taiwan, April 2020.

[21] Julian Bruyat. Web assembly pour le web sémantique. Master's thesis, Université Claude Bernard Lyon 1, 2020. http://bruy.at/BRUYAT_Rapport_WasmPourWebSem.pdf.

[22] Mehdi Terdjimi, Lionel Médini, and Michael Mrissa. Hylar: Hybrid location-agnostic reasoning. In *ESWC Developers Workshop 2015*, page 1, 2015.

[23] Julien Subercaze, Christophe Gravier, Jules Chevalier, and Frederique Laforest. Inferray: fast in-memory rdf inference. 2016.

[24] Charles L Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. In *Readings in Artificial Intelligence and Databases*, pages 547–559. Elsevier, 1989.

[25] Ashish Gupta, Inderpal Singh Mumick, and Venkatramanan Siva Subrahmanian. Maintaining views incrementally. *ACM SIGMOD Record*, 22(2):157–166, 1993.

[26] Jacopo Urbani, Alessandro Margara, Ceriel Jacobs, Frank Van Harmelen, and Henri Bal. Dynamite: Parallel materialization of dynamic rdf data. In *International Semantic Web Conference*, pages 657–672. Springer, 2013.

[27] Yavor Nenov, Robert Piro, Boris Motik, Ian Horrocks, Zhe Wu, and Jay Banerjee. Rdfox: A highly-scalable rdf store. In *International Semantic Web Conference*, pages 3–20. Springer, 2015.

[28] Mehdi Terdjimi, Lionel Médini, and Michael Mrissa. Web reasoning using fact tagging. In *Companion Proceedings of the The Web Conference 2018*, pages 1587–1594, 2018.

[29] Daniele Dell'Aglio, Emanuele Della Valle, Frank van Harmelen, and Abraham Bernstein. Stream reasoning: A survey and outlook. *Data Science*, 1(1-2):59–83, 2017.

[30] Jules Chevalier, Julien Subercaze, Christophe Gravier, and Frédérique Laforest. Incremental and directed rule-based inference on rdfs. In *International Conference on Database and Expert Systems Applications*, pages 287–294. Springer, 2016.

[31] Daniele Dell'Aglio and Emanuele Della Valle. Incremental reasoning on rdf streams., 2014.

[32] Martin Peters, Christopher Brink, Sabine Sachweh, and Albert Zündorf. Rule-based reasoning on massively parallel hardware. In *SSWS@ ISWC*, pages 33–49, 2013.

[33] Martin Peters, Christopher Brink, Sabine Sachweh, and Albert Zündorf. Scaling parallel rule-based reasoning. In *European Semantic Web Conference*, pages 270–285. Springer, 2014.

[34] Pramod J Sadalage and Martin Fowler. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education, 2013.

[35] Fausto Giunchiglia and Chiara Ghidini. Local models semantics, or contextual reasoning= locality+ compatibility. *KR*, 98:282–289, 1998.

[36] Jérémy Lhez, Chan Le Duc, Thinh Dong, and Myriam Lamolle. Decentralized reasoning on a network of aligned ontologies with link keys. In *International Semantic Web Conference*, pages 418–434. Springer, 2019.

[37] Thorsten Liebig, Marko Luther, Olaf Noppens, and Michael Wessel. Owllink. *Semantic Web – Interoperability, Usability, Applicability*, 2(1):23–32, 2011.

[38] Rob Shearer, Boris Motik, and Ian Horrocks. Hermit: A highly-efficient owl reasoner. In *Owled*, volume 432, page 91, 2008.

[39] Nicolas Seydoux, Khalil Drira, Nathalie Hernandez, and Thierry Monteil. Edr: A generic approach for the dynamic distribution of rule-based reasoning in a cloud-fog continuum. *Semantic Web Journal*, 2019.

[40] A.A. Romero, B.C. Grau, I. Horrocks, and Ernesto Jiménez-Ruiz. More: A modular owl reasoner for ontology classification. *CEUR Workshop Proceedings*, 1015:68–74, 01 2013.