A Linked Data Reasoner in the Cloud

Jules Chevalier
jules.chevalier@univ-st-etienne.fr

LT2C, Télécom Saint Etienne, Université Jean Monnet

février 2014

Superviseurs :
Frédérique Laforest
Christophe Gravier
Julien Subercaze

Les données du Web

- En 2012 : 90% des données du monde générées en 2 ans
- Nous produisons chaque jour plus de données que depuis le début de l'humanité
- Chaque minutes :
  - 571 nouveaux sites web
  - 277.000 tweets
  - 2.000.000 de recherches Google
  - 72h de vidéos uploadées sur YouTube
  - 350Go de données traitées par Facebook

source : *How big is BIG DATA, http ://removeandreplace.com*

Le Web Sémantique

- Permet de formaliser des concepts
- Vise à standardiser ces représentations
- Crée des liens entre les données
- Permet des traitements automatiques sur ces données
  - Vérification de l'intégrité
  - Extraction de connaissances implicites
  - Requêtage de la base de connaissance

- Chaque connaissance est un triple : `<Homer, father, Bart>`
- Une ontologie est un ensemble de triples représentant un domaine, un concept

  Exemple : DBPedia regroupe des connaissances extraites de Wikipedia

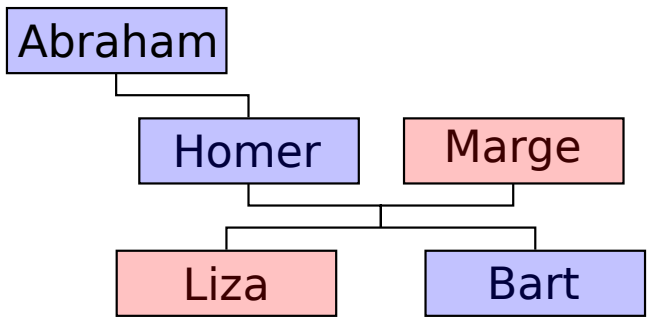
Le Web Sémantique

- Permet de formaliser des concepts
- Vise à standardiser ces représentations
- Crée des liens entre les données
- Permet des traitements automatiques sur ces données
  - Vérification de l'intégrité
  - **Extraction de connaissances implicites => Raisonnement**
  - Requêtage de la base de connaissance

- Chaque connaissance est un triple : `<Homer, father, Bart>`
- Une ontologie est un ensemble de triples représentant un domaine, un concept

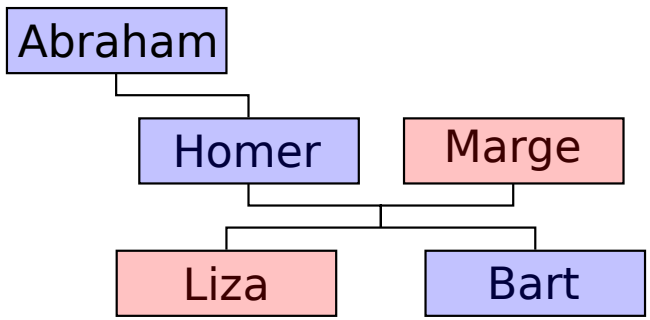
  Exemple : DBPedia regroupe des connaissances extraites de Wikipedia

Raisonnement



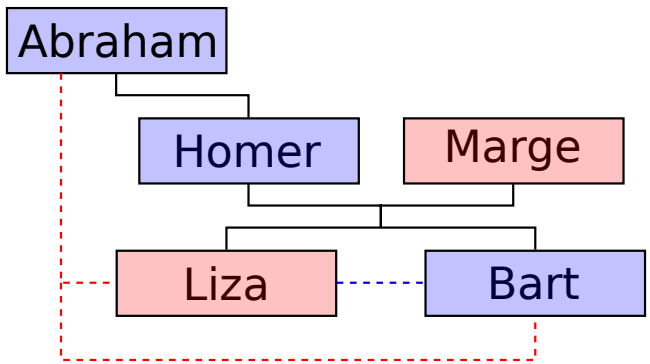
- Base de connaissance :
  - Abraham father Homer
  - Homer father Liza
  - Homer father Bart
  - Marge mother Liza
  - Marge mother bart

Raisonnement



- Base de connaissance :
  - Abraham father Homer
  - Homer father Liza
  - Homer father Bart
  - Marge mother Liza
  - Marge mother bart
- Règles :
  - X father Y
Y father Z $\Rightarrow$ X grandfather Z
  - X father Y
X father Z $\Rightarrow$ Y sibling Z

Raisonnement



- Base de connaissance :
  - Abraham father Homer
  - Homer father Liza
  - Homer father Bart
  - Marge mother Liza
  - Marge mother bart
- Règles :
  - X father Y
Y father Z $\Rightarrow$ X grandfather Z
  - X father Y
X father Z $\Rightarrow$ Y sibling Z
- Raisonnement :
  - Abraham grandfather Liza
  - Abraham grandfather Bart
  - Bart sibling Liza
  - Liza sibling Bart

Fragments

- Un fragment est un ensemble de règles
- Les standards du Web Sématique proposent différents fragments pré-définis (RDFS, OWL Lite, OWL DL, OWL Full, ...)
- Plus un fragment est expressif, plus le raisonnement est coûteux
- Choisir un fragment est un compromis entre expressivité et complexité calculatoire

- La complexité d'un fragment peut être représentée par le graphe de dépendances de ses règles

Sujet

- On sait raisonner sur des petites bases de connaissances

Problématique

- Comment raisonner sur à l'échelle du Web

Sujet

- On sait raisonner sur des petites bases de connaissances

Problématique

- Comment raisonner sur à l'échelle du Web

Idées

- Distribuer le processus de raisonnement sur plusieurs nœuds
- Utiliser le Cloud comme environnement de déployement
  - Flexibilité : Adaptation nombre/puissance des VM
  - Limitation du coût : Seul ce qui est utilisé est facturé
  - Réactivité : Faible latence entre les nœuds

Sujet

Objectifs

- Raisonner sur de larges ontologies
- Indépendamment du fragment
- En intégrant certaines contraintes (temps/quantité de triples générée)

Sujet

Objectifs

- Raisonner sur de larges ontologies
- Indépendamment du fragment
- En intégrant certaines contraintes (temps/quantité de triples générée)

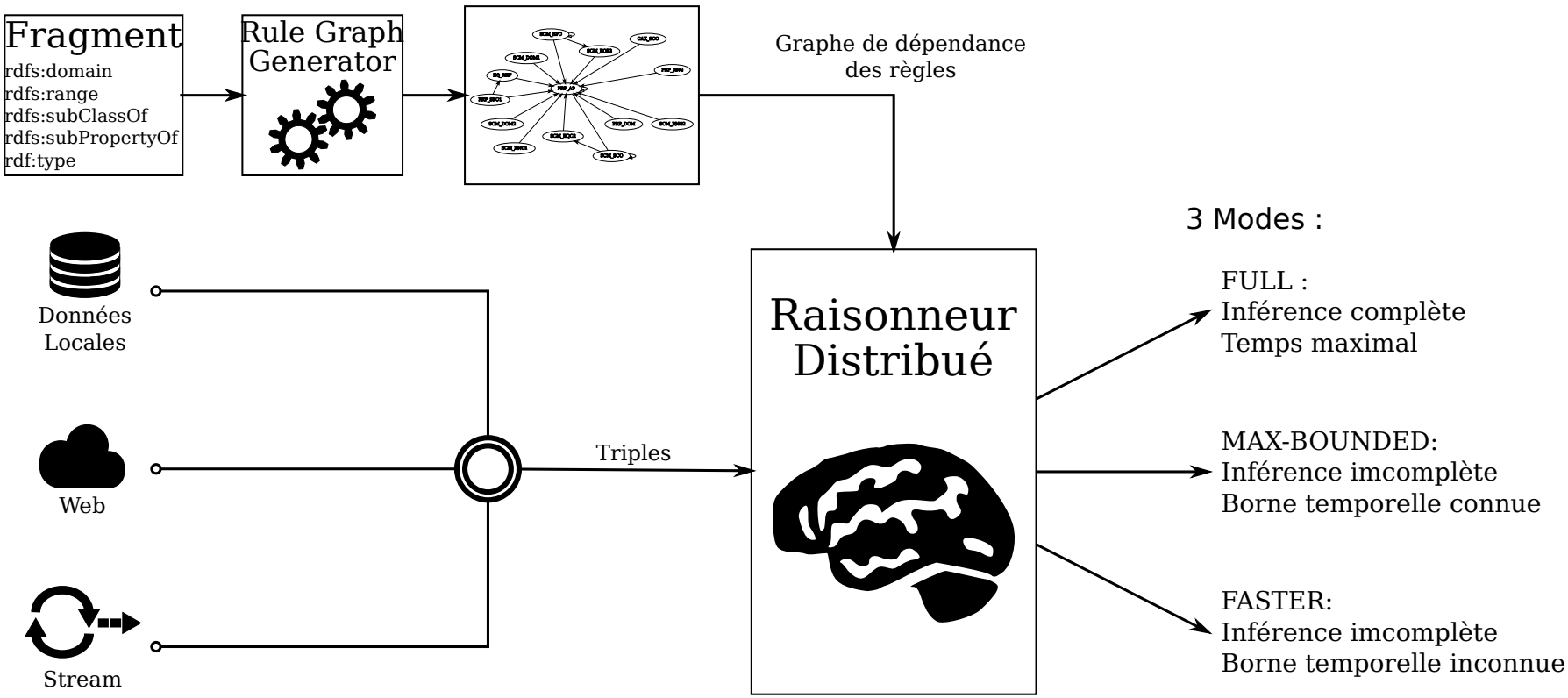
Verrous

- Les règles forment un graphe de dépendance cyclique
- La quantité imprédictible de triples générés
- La complexité dépendante du fragment et des données

Grandes lignes de notre solution

- Accès des données par tous les nœuds
- Structures de données dédiées au raisonnement distribué
- Ordonnancement intelligent des règles
- 3 modes de fonctionnement : `Full`, `Max-Bounded`, `Faster`

Fonctionnement général



Structures de données pour le raisonnement distribué

Numbers Everyone Should Know

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

source : *Software Engineering Advice from Building Large-Scale Distributed Systems*, Jeff Dean

Structures de données pour le raisonnement distribué

Numbers Everyone Should Know

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	**10,000,000 ns**
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

source : *Software Engineering Advice from Building Large-Scale Distributed Systems*, Jeff Dean

Structures de données pour le raisonnement distribué

- Besoin de structures de données
  - Concurrentes
  - Indexées
  - En RAM
- Supportant
  - Le partage via le réseau
  - Un accès disque minimisé (seulement pour le chargement)

Structures de données pour le raisonnement distribué

- Besoin de structures de données
  - Concurrentes
  - Indexées
  - En RAM
- Supportant
  - Le partage via le réseau
  - Un accès disque minimisé (seulement pour le chargement)

Avancement actuel

- Le TripleStore est un objet partagé par tous les threads
  - Concurrent
  - Indexé
  - Triples Immutable
- Chaque instance de règle peut y accéder
- Cet accès direct limite les duplicatas

Indépendance au fragment

- Fragment (ensemble de règles) personnalisable
- Possibilité de définir de nouvelles règles
- Optimisations adaptées à chaque règle
- Système de recommandation de fragments

Indépendance au fragment

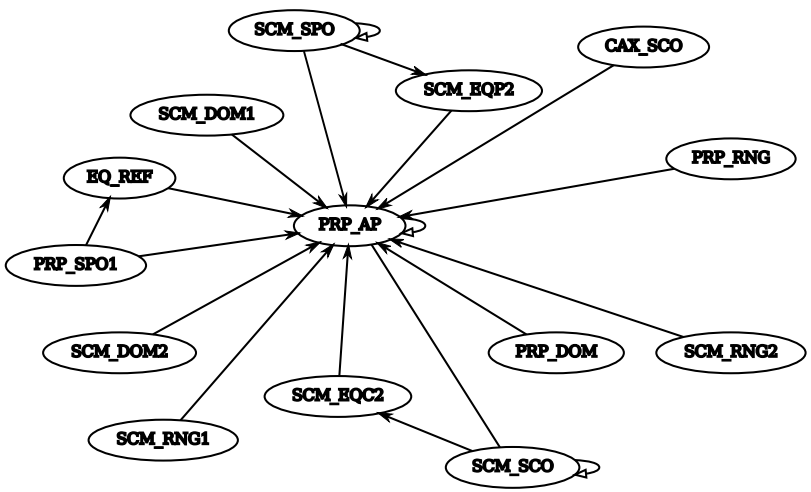
- Fragment (ensemble de règles) personnalisable
- Possibilité de définir de nouvelles règles
- Optimisations adaptées à chaque règle
- Système de recommandation de fragments

Avancement actuel

- Chaque règle est une classe distincte
- Ordonnancement dynamique
- Choix du fragment simplifié

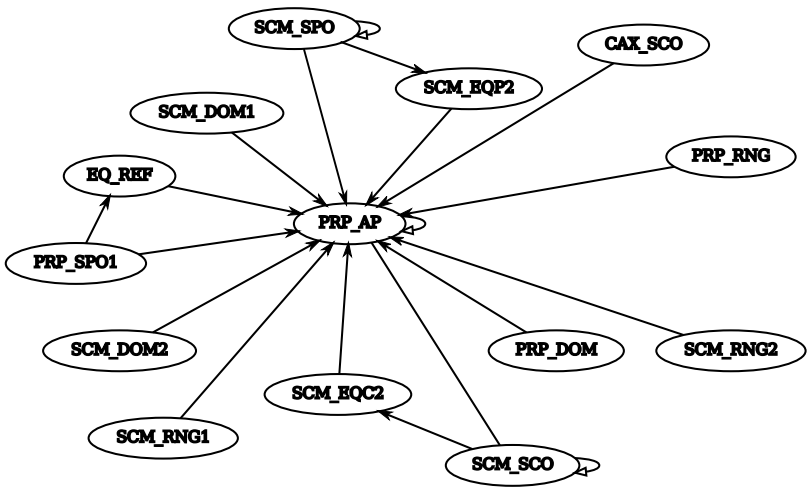
Ordonnancement adaptatif intelligent

- Ordonnancement automatique des règles
- Calculé grâce au graphe de dépendance
- Capable de s'adapter pendant l'inférence



Ordonnancement adaptatif intelligent

- Ordonnancement automatique des règles
- Calculé grâce au graphe de dépendance
- Capable de s'adapter pendant l'inférence



Avancement actuel

- Ordonnancement des règles au lancement
- Ne sont traitées que les règles du fragment choisit

Fonctionnement sous forme de flux

- La gestion de flux permet de :
  - Prévenir l'attente des nœuds
  - Adapter le réseau des nœuds
  - Éviter l'isolation des données
- Convient parfaitement au traitement en streaming
  - Batch $\sqsubseteq$ Stream
  - Mais Stream $\not\sqsubseteq$ Batch ! (En particulier pour `Max-Bounded` et `Faster`)

Fonctionnement sous forme de flux

- La gestion de flux permet de :
  - Prévenir l'attente des nœuds
  - Adapter le réseau des nœuds
  - Éviter l'isolation des données
- Convient parfaitement au traitement en streaming
  - Batch $\sqsubseteq$ Stream
  - Mais Stream $\not\sqsubseteq$ Batch ! (En particulier pour `Max-Bounded` et `Faster`)

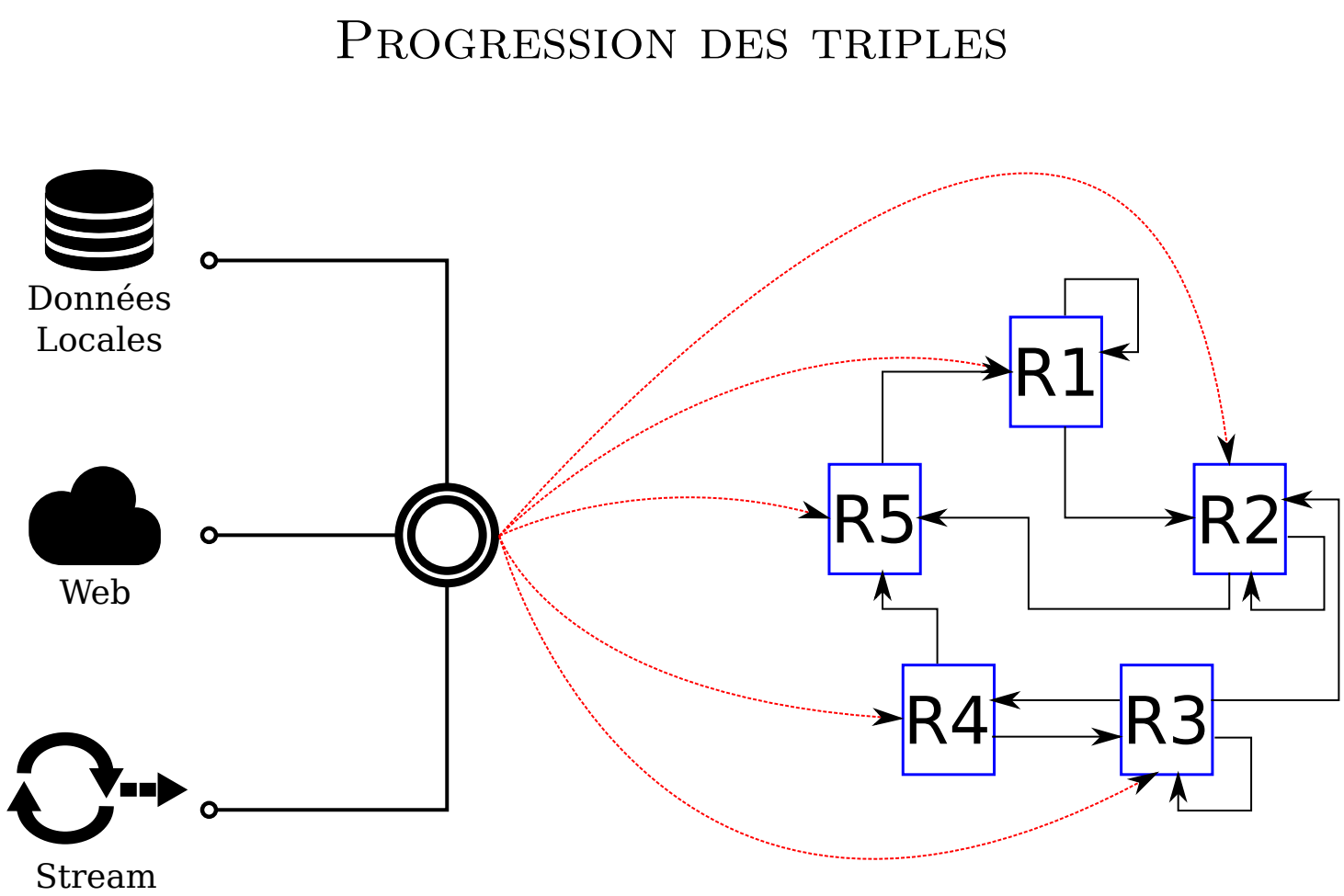
Avancement actuel

- Les données en entrées sont en lot ou streamées
- Les règles fonctionnent par flux
- Plusieurs instances d'une même règle peuvent tourner en même temps

Version multithreaded

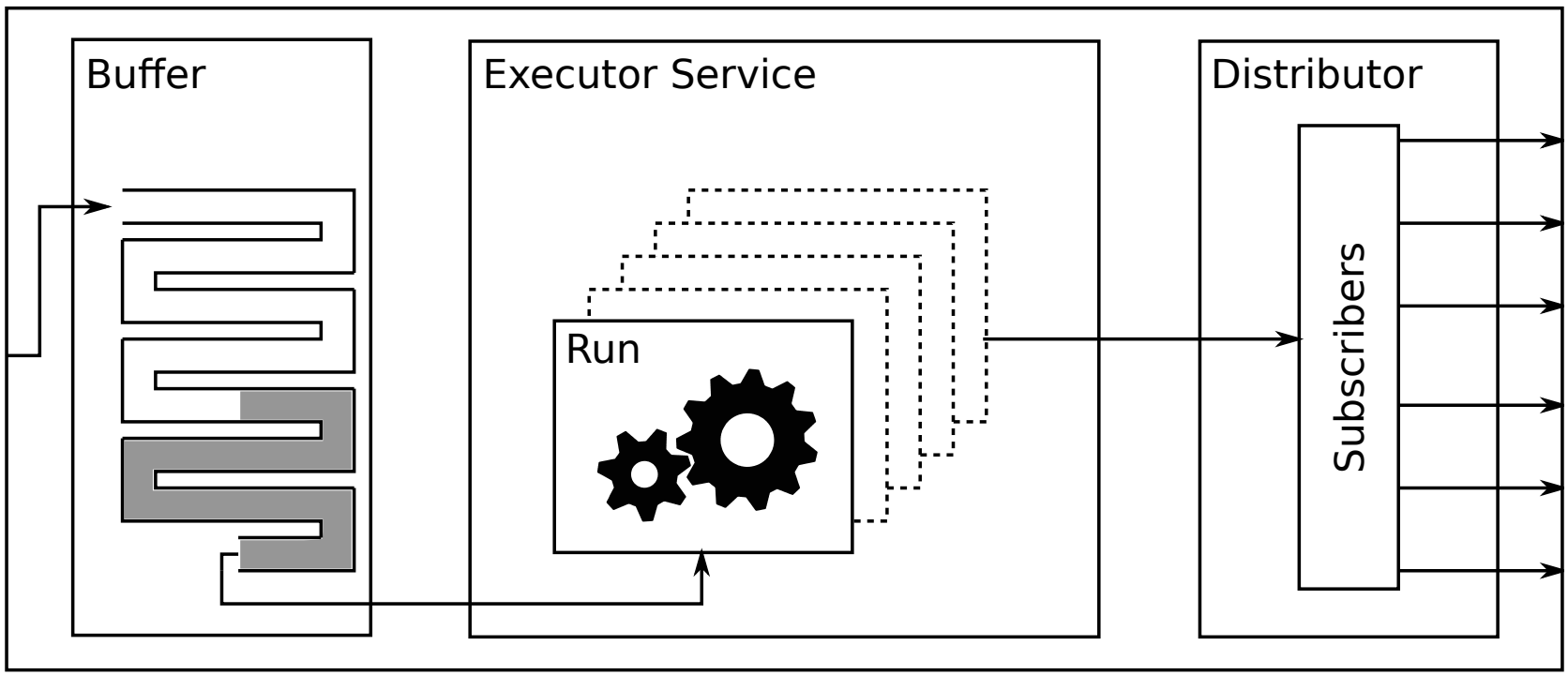
- Avant de développer une version distribuée, nous optimisons une version multithread, pour la migrer ensuite vers une architecture distribuée.

Architecture



Architecture

Fonctionnement des règles



Conclusion

Objectifs

- Raisonner sur de larges ontologies
- Indépendamment du fragment
- En intégrant certaines contraintes (temps/quantité de triples générée)

Conclusion

Avantages de notre solution

- Un nœud n'est pas dédié à une tâche ou une règle
- Ordonnancement automatique et adaptatif des règles
- Données partagées en mémoire
- Traitement en streaming

Prévisions

- Février 2014 - Version stable du raisonneur multithread
  - Test de la 1ere version contre les solutions existantes
- Mars 2014 - Implémentation de la version distribuée dans le Cloud
  - Test de la version distribuée contre la version multithread
- Mai 2014 - Première implémentation de l'ordonnancement intelligent
  - Test de cette version contre la version précédente
- Novembre 2014 - Raisonneur open-sourcé et documenté
  - Batterie de tests pour les 3 problèmes adressés
- Janvier 2015 - Fonctions supplémentaires :
  - Assister l'utilisateur pour le choix du fragment
  - Propositions d'améliorations du fragment
  - Faire des prédictions sur l'inférence (temps et quantité)
  - Optimisations du raisonneur lui-même
- Avril 2015 - Rédaction du manuscrit
- Automne 2015 - Soutenance de la thèse

Merci de votre attention

Questions ?