UNIVERSITÉ PARIS-SUD

ÉCOLE DOCTORALE INFORMATIQUE DE PARIS-SUD (ED 427)

Laboratoire de Recherche en Informatique (LRI)

***DISCIPLINE** INFORMATIQUE*

SYNTHÈSE DE LA THÈSE DE DOCTORAT

presentée en vue d'obtention du titre de docteur le 25/09/2014

par **Jesús CAMACHO RODRÍGUEZ**

Techniques efficaces de gestion de données Web à grande échelle

Thèse dirigée par :

Dario Colazzo	Université Paris-Dauphine
Ioana Manolescu	Inria and Université Paris-Sud

Rapporteurs :

Donald Kossmann	ETH Zürich, Switzerland
Volker Markl	TU Berlin, Germany

Examinateurs :

Reza Akbarinia	Inria and Université Montpellier II
Marc Baboulin	Université Paris-Sud and Inria
Philippe Rigaux	Conservatoire National des Arts et Métiers

Introduction

Depuis l'apparition de la toile mondiale (le Web), le volume et la vitesse de création des données sont en plein essor. Les sites internet regroupant des données volumineuses comme les catalogues de produits, les sites de médias sociaux, RSS et tweets, blogs et publications en ligne illustrent cette tendance. De plus les données du Web sont très hétérogènes et sont produites dans divers formats (parfois complexes).

Par conséquent, une partie croissante des données intéressantes du monde sont partagées sur le Web ou directement produites par et pour des plateformes Web. De nos jours de nombreuses organisations reconnaissent la valeur de la richesse des données Web. Cependant, le stockage et le traitement de ces importants volumes de données hétérogènes (structurées, semi-structurées et non structurées) rencontrent un certain nombre de défis concernant : les modèles adéquats pour décrire des données hétérogènes, des structures complexes, des données distribuées ; les langages devant être utilisés pour manipuler les données Web de façon expressive ; et enfin, l'architecture et les algorithmes concrets à utiliser pour mettre en œuvre les langages choisis de façon efficace.

Le travail accompli dans cette thèse s'inscrit dans une perspective visant à utiliser les infrastructures *cloud* et le *parallélisme massif* des structures numériques afin de répondre aux défis sus-cités. Nous justifions ces choix ci-dessous avant d'exposer nos contributions.

Le développement récent des offres commerciales autour du cloud computing [AWS, GCP, WA] a fortement impacté la recherche et le développement des plateformes de distribution numérique. Les fournisseurs de cloud proposent une infrastructure de distribution extensible qui peut être utilisée pour le stockage et le traitement des données. *Dans cette thèse nous étudierons des architectures efficaces de gestion des données Web construites sur ces plateformes* et nous nous intéresserons aux performances des plateformes pour l'entreposage de données Web. De plus, la consommation des ressources est directement traduite en coûts financiers pour l'utilisateur exploitant la plateforme pour l'entreposage des données, il devient donc nécessaire d'étudier l'interaction entre les différentes techniques classiques de gestion des données (plus particulièrement l'indexation de contenu) et ces coûts.

En parallèle du développement des plateformes cloud, les modèles de programmation qui parallélisent harmonieusement le traitement de grandes quantités de données en les distribuant sur un ensemble de machines standards ont fait l'objet d'un intérêt considérable, le modèle MapReduce [DG04] à l'origine de ceci est bien connu ; il existe désormais divers frameworks plus récents et expressifs tels que [BEH+10, ZCD+12]. Ces modèles étant de plus en plus utilisés pour réaliser les tâches de traitement de données analytiques, le besoin d'un langage plus performant permettant de simplifier l'écriture de requêtes complexes pour ces systèmes se fait donc sentir. *Dans cette thèse, nous étudierons des techniques efficaces permettant de traduire des opérations complexes exprimées dans ces langages élaborés par des modèles de programmation parallèles* comme ceux sus-cités. Nous nous intéresserons également à *diverses opportunités d'optimisation associées à ces nouveaux systèmes*.

Le but de cette thèse étant d'étudier des techniques efficaces pour la gestion de grandes quantités de données Web, elle s'intéresse à trois problèmes majeurs : la gestion de données Web par l'utilisation de services commerciaux cloud, la parallélisation du langage XQuery sur des infrastructures à grande échelle et l'optimisation de la réutilisation des requêtes Pig Latin. Nous présentons ici un aperçu de chaque problème et exposons nos principales contributions.

AMADA : entreposage de données Web dans le cloud

Au cours des dernières années, les grandes compagnies des TI comme Amazon, Google ou Microsoft ont commencé à fournir un nombre croissant de services cloud construits sur leurs infrastructures. En utilisant ces plateformes commerciales cloud, les organisations et les individus peuvent bénéficier d'infrastructures largement développées et s'en servir pour créer leurs applications. L'élasticité de ces plateformes est un élément important, c'est-à-dire la capacité à attribuer plus (ou moins) de puissance informatique, d'espace de stockage ou d'autres services selon que la demande pour l'application est croissante ou décroissante. Les services cloud sont loués selon un contrat de service (SLA) caractérisant leur performance, fiabilité, etc.

Bien que les services offerts par les clouds publics soient variables, ils proposent tous une forme de stockage évolutive, durable et facilement disponible pour les fichiers ou des objets binaires volumineux (équivalents). Les plateformes cloud proposent également des machines virtuelles (appelées des *instances*) qui s'activent ou se désactivent selon les besoins et sur lesquelles il est possible d'exécuter des codes. AMADA opère comme Logiciel en tant que Service (SaaS, Software as a Service), permettant d'entreposer de grandes quantités de données Web dans le cloud :

— Pour *stocker les données*, pour les télécharger dans le fichier de stockage cloud.
— Pour *traiter une requête*, pour lancer des instances, leur faire lire les données depuis le fichier de stockage, calculer les résultats des requêtes et recevoir les résultats.

Á l'évidence, la performance (temps de réponse) entraînée par ce processus revêt une certaine importance ; toutefois, des coûts financiers sont associés à ce scénario. Ce sont les coûts de téléchargement, de stockage et de traitement des données pour répondre à la requête. Les coûts facturés par le fournisseur de cloud sont fonction de l'effort total (ou du travail total), en d'autres termes il s'agit de la consommation totale de toutes les ressources cloud occasionnées par le stockage et le traitement de la requête. Plus précisément, lorsque l'entreposage des données est volumineux, si l'évaluation d'une requête implique toutes (ou une grande partie) des données, cela entraînera des coûts élevés pour : (i) lire les données depuis le fichier de stockage et (ii) traiter la requête sur ces données.

Dans ce travail nous avons exploré l'utilisation de *l'indexation de contenu* en tant qu'outil d'amélioration des performances des requêtes et de réduction des coûts totaux liés à l'exploitation de l'entrepôt de données basé sur le cloud.

Nous nous sommes concentrés sur une structure de données arborescente et en particulier du XML en raison de la popularité de ce langage, ainsi que sur des formats

arborescents tels que JSON. Nous avons pris comme exemple la plateforme Amazon Web Services (AWS) qui fait partie des plus populaires et qui a déjà été utilisée comme exemple dans des travaux de recherche [BFG+08, SDQR10]. Etant donné qu'il existe de fortes ressemblances entre les plateformes commerciales cloud, nos résultats sont facilement transférables à d'autres plateformes.

Contributions. Les contributions de ce travail sont les suivantes.

— Nous présentons une architecture générique pour l'entreposage à grande échelle de données Web complexes en utilisant les plateformes commerciales cloud. Une attention particulière est apportée à la modélisation des coûts monétaires associés à l'exploitation de l'entrepôt de données.

— Nous étudions l'utilisation de l'indexation de contenu pour des données arborescentes (en particulier des données XML). Notre travail a principalement ciblé la construction et l'exploitation de différentes stratégies d'indexation afin d'améliorer à la fois la vitesse de traitement et de réduire la consommation des ressources cloud (et par conséquent les coûts opérationnels de l'entreposage des données).

— Nous avons décrit l'exécution concrète de notre architecture sur la plateforme Amazon Web Services (AWS), l'une des plateformes commerciales cloud les plus populaires de nos jours. Nous avons montré que l'indexation permet de réduire d'environ deux-tiers le temps de traitement et les coûts d'un facteur dix. De plus les coûts de création d'indexation sont très rapidement amortis car la quantité de requêtes traitées est plus importante. Etant donné qu'il existe de fortes ressemblances entre les plateformes commerciales cloud, nos résultats sont facilement transférables à d'autres plateformes.

Diffusion des connaissances. Une version préliminaire de ce travail a été présentée lors d'un séminaire [CRCM12], le matériel de cette thèse quant à lui se rapporte à une publication de conférence internationale [CRCM13]. La gestion de données RDF en utilisant AMADA a été étudiée en parallèle par notre équipe et représente une partie conséquente de ce travail de thèse [BCRG+14]. Le système AMADA a été présenté en [AABCR+12] et a été placé en open source en mars 2013 [1].

PAXQuery : traitement parallèle efficace de XQuery

Etant de loin le framework *parallèle implicite* le plus largement adopté, MapReduce [DG04] est un modèle de traitement très simple consistant en deux fonctions, *Map* qui distribue le traitement sur différents couples (clé, valeur) et *Reduce* qui associe toutes les valeurs calculées par *Map* pour chaque clé distincte. Bien que la simplicité de MapReduce soit un avantage, elle constitue également une restriction car le traitement de données volumineuses est assuré par des programmes complexes consistant en des fonctions *Map* et *Reduce*. Plus précisément, étant donné que ces tâches sont conceptuellement basiques, il faut souvent écrire des programmes comprenant différentes tâches

1. http://cloak.saclay.inria.fr/research/amada/

de traitement successives, ce qui limite le parallélisme. Pour surmonter ce problème, des abstractions plus puissantes ont été proposées afin de traiter massivement en parallèle des données complexes, telles que les *Resilient Distributed Datasets* [ZCD+12] ou le modèle de programmation *PArallelization ConTracts* [BEH+10] (ou PACT).

Les programmes PACT sont d'une nature suffisamment déclarative pour mettre à profit les techniques de compilation intelligente afin d'être évalués de façon très efficace. Durant la compilation, le compilateur choisit une stratégie (plan) optimale qui permet de maximiser les opportunités de parallélisation et donc la performance.

En résumé, les PACT généralisent MapReduce en (i) manipulant les registres avec un nombre illimité de champs, à l'inverse des couple (clé, valeur), (ii) autorisent la définition d'opérateurs parallèles au moyen de fonctions du second degré et (iii) permettent qu'un opérateur parallèle reçoive les résultats d'un certain nombre d'autres opérateurs en tant qu'entrées. La programmation PACT fait partie de la plateforme open source **Stratosphere** [Str] qui travaille sur Hadoop Distributed File System (HDFS) [Had].

Dans ce travail nous présenterons PAXQuery, un processeur massivement parallèle de requêtes XML. Á partir d'un important ensemble de documents XML, évaluer une requête qui *navigue sur ces documents et présente les résultats de différents documents* constitue un véritable défi de performance qui peut être traité par le parallélisme. S'inspirant d'autres langages analytiques de haut niveau compilés par les programmes MapReduce (par exemple Pig Latin [ORS+08], Hive [TSJ+10] ou Jaql [BEG+11]), PAXQuery permet de traduire les requêtes XML en plans PACT. Le principal avantage de cette approche est le *parallélisme implicite* : il n'est pas nécessaire que l'application ou l'utilisateur partitionne l'entrée XML ou la requête sur les différents nœuds. Ceci se démarque des travaux précédents [BCM+13, CLK+12, KCS11]. De plus nous pouvons compter sur la plateforme Stratosphere pour optimiser le plan PACT et le transformer automatiquement en flux de données évalué en parallèle sur Hadoop ; ces étapes sont expliquées dans [BEH+10].

Contributions. Les contributions de ce travail sont les suivantes.

— Nous présentons une nouvelle *méthodologie pour l'évaluation massivement parallèle de XQuery*, basée sur la programmation PACT et des recherches antérieures sur l'optimisation algébrique de XQuery.

— Nous apportons un *algorithme de traduction* pour transformer les opérateurs algébriques requis par un important volume de XQuery en opérateurs du framework parallèle PACT. Ceci permet une évaluation parallèle XQuery *sans avoir à demander à l'application de fournir des efforts de partitionnement des données ou des requêtes*.

Afin d'atteindre cet objectif, nous avons tout d'abord traduit les instances de données XML (arbre avec réseaux de nœuds) en registres emboîtés PACT, afin de s'assurer que les résultats des requêtes XML sont transmis après la manipulation PACT des registres emboîtés.

Deuxièmement, nous avons comblé le fossé entre l'algèbre XQuery – et plus particulièrement plusieurs types de liens [DPX04, MPV09, MHM06] allant au-delà des simples liens cumulatifs – et les opérateurs PACT (tels que MapRe-

duce) qui dépendent fortement du regroupement des données d'entrée en ce qui concerne l'égalité des clés.
L'intérêt de notre traduction des liens complexes en PACT va au-delà du contexte XQuery car elle pourrait permettre de compiler d'autres langages de haut niveau [BEG+11, ORS+08, TSJ+10] en PACT et profiter ainsi de sa performance.

— Nous avons pleinement mis en œuvre notre technique de traduction sur notre plateforme PAXQuery. Nous présentons des expériences démontrant que notre approche de traduction (*i*) permet de paralléliser efficacement l'évaluation XQuery en profitant des avantages du framework PACT, et (*ii*) s'étend bien au-delà des approches alternatives des évaluations XQuery implicitement parallèles, en particulier dès que des liens entre les documents existent dans la tâche.

Diffusion des connaissances. Une courte présentation du système PAXQuery a été publiée lors du séminaire [CRCM14]. Le matériel présenté dans cette thèse est proposé pour la publication dans une revue internationale. Actuellement, PAXQuery est en train d'être intégré à la dernière version de Stratosphere et sera placé en open source en tant qu'extension de la plateforme Stratosphere dans un futur proche [2].

Optimisation de la réutilisation pour le Pig Latin

L'efficacité du traitement d'un important volume de données s'est récemment orientée vers les modèles de traitement massivement parallèles tels que MapReduce qui est le plus populaire. Toutefois, la simplicité du modèle MapReduce amène à développer des programmes relativement complexes permettant de traiter des tâches même modérément complexes. Afin de faciliter la spécification des tâches de traitement des données selon le mode massivement parallèle, un certain nombre de langages de requêtes de haut niveau plus faciles d'utilisation ont été développés et sont automatiquement compilés en programmes MapReduce. Les langages largement utilisés sont le Pig Latin [ORS+08], HiveQL [TSJ+10] ou Jaql [BEG+11].

Dans ce travail nous avons utilisé le langage Pig Latin qui a suscité un intérêt considérable chez les développeurs d'applications et au sein de la recherche. Le Pig Latin permet d'exprimer des tâches de traitement de données analytiques complexes sous la forme d'un flux de données primitif. Les programmes Pig Latin (également appelés *scripts*) sont automatiquement optimisés et compilés en tâches MapReduce par le système Apache Pig [Pig].

Dans un jeu typique de scripts Pig Latin, des sous-expressions peuvent être identiques (ou équivalentes), c'est-à-dire : des fragments de script appliquant le même traitement sur les mêmes entrées mais apparaissant à des endroits différents du même ou de plusieurs scripts. Bien que le moteur Pig Latin comporte un optimisateur de requête, il ne lui est actuellement pas possible de repérer la répétition de sous-expressions. Par

2. http://cloak.saclay.inria.fr/research/paxquery/

conséquent elles seront exécutées autant de fois qu'elles apparaissent dans le jeu de scripts Pig Latin bien qu'il y ait là une opportunité évidente d'optimiser les performances en identifiant les sous-expressions communes, en ne les exécutant qu'à une seule reprise et en réutilisant les résultats des calculs de chaque script qui les utilise.

L'objectif du présent travail est *d'identifier et de réutiliser automatiquement les sous-expressions communes présentent dans les scripts Pig Latin*. Le problème est à l'évidence similaire aux problèmes d'optimisation multi-requêtes et de réutilisation des tâches ; toutefois les paramètres Pig Latin ont permis d'aborder de nouveaux aspects du problème, ce qui nous permet de proposer des algorithmes spécialisés pour les résoudre.

Exemple. Un script Pig Latin est un ensemble d'expressions liantes (*binding expressions*) et d'expressions de stockage (*store expressions*). Chaque expression liante est suivie de la syntaxe var = op, signifiant que l'expression op sera évaluée et que le multi-ensemble de tuples générés sera lié à la variable var. Par la suite, var peut être utilisé par des expressions de rappel dans le script.

Prenons le script Pig Latin a_1 suivant :

```
1 A = LOAD 'page_views' AS (user, time, www);
2 B = LOAD 'users' AS (name, zip);
3 R = JOIN A BY user, B BY name;
4 S = FOREACH R GENERATE user, time, zip;
5 STORE S INTO 'a1out1';
6 T = JOIN A BY user LEFT, B BY name;
7 STORE T INTO 'a1out2';
```

La ligne 1 permet de charger les données depuis le fichier `page_views` et crée un multi-ensemble de tuples liés à la variable `A`. Chacun de ces tuples est constitué de trois attributs `(user,time,www)`. La ligne 2 permet de charger un second fichier et lie le multi-ensemble de tuples résultants à `B`. La ligne 3 lie les tuples de `A` et de `B` d'après l'égalité des valeurs liées aux attributs `user` et `name`. La ligne suivante utilise l'important opérateur Pig Latin `FOREACH` qui permet d'appliquer une fonction à chaque tuple du multi-ensemble d'entrée. Dans le cas présent, la ligne 4 projette les attributs `user`, `time` et `zip` sur chaque tuple de `C`. Le résultat est ensuite stocké dans le fichier `a1out1`. La ligne 6 quant à elle permet d'exécuter un lien extérieur gauche sur les tuples de `A` et `B`, d'après l'égalité des valeurs liées aux attributs `user` et `name`, le résultat est stocké dans le fichier `a1out2`.

Le script suivant a_2 n'exécute qu'un lien extérieur gauche sur les mêmes entrées :

```
1 A = LOAD 'page_views' AS (user, time, www);
2 B = LOAD 'users' AS (name, zip);
3 R = JOIN A BY user LEFT, B BY name;
4 STORE R INTO 'a2out';
```

Le script b que nous introduisons par la suite produit les mêmes résultats que a_1 et a_2 :

```
1 A = LOAD 'page_views' AS (user, time, www);
2 B = LOAD 'users' AS (name, zip);
3 R = COGROUP A BY user, B BY name;
4 S = FOREACH R GENERATE flatten(A), flatten(B);
5 T = FOREACH S GENERATE user, time, zip;
6 STORE T INTO 'a1out1';
7 U = FOREACH R GENERATE flatten(A),
8                  flatten (isEmpty(B) ? {(null,null,null)} : B);
9 STORE U INTO 'a1out2';
10 STORE U INTO 'a2out';
```

Cependant, *le temps d'exécution de b est de* 45% *par rapport aux temps d'exécution combinés de* a_1 *et* a_2. La raison à ceci est double. Tout d'abord il faut observer que les liens sont réécrits dans une opération `COGROUP` [3] (ligne 3) et des opérations `FOREACH` (lignes 4 et 7-8). L'intérêt d'utiliser cogroup est qu'il est possible par une restructuration simple de faire ressortir du résultat du cogroup divers types de liens (naturel, extérieur, emboîté, semi-lié etc.). Cette opération de restructuration varie selon que l'on souhaite générer un lien entre `A` et `B` comme dans le script a_1 (ligne 4) ou un lien extérieur gauche entre `A` et `B` comme dans les scripts a_1 et a_2 (lignes 7-8). Par conséquent la raison première pour laquelle b est plus efficace que a_1 et a_2 est que la sortie `COGROUP` est réutilisée pour générer le résultat des deux liens. La seconde raison est que dans b, le lien extérieur gauche n'est calculé qu'une seule fois et que son résultat est utilisé pour obtenir le résultat voulu par le script a_1 (ligne 9) et a_2 (ligne 10).

Contributions. Les contributions de ce travail sont les suivantes.

— Nous avons formalisé la représentation des scripts Pig Latin d'après le formalisme algébrique bien établi existant, plus précisément le *Nested Relational Algebra for Bags* (NRAB) [GM93]. Ceci permet de travailler à partir d'une base formelle et permet d'identifier avec précision les expressions communes dans les ensembles de scripts Pig Latin.

— Nous proposons PigReuse, un algorithme d'optimisation multi-requêtes qui permet de combiner des sous-expressions équivalentes qu'il identifie dans un Graph Orienté Acyclique (DAG) des opérateurs NRAB, correspondant à l'ensemble de scripts Pig Latin. Après avoir identifié de telles opportunités de réutilisation, PigReuse produit un plan de fusion optimal où les calculs redondants seront éliminés. PigReuse se base sur un solveur de Programmation Linéaire en Variables Binaires efficace sélectionnant les meilleurs plans selon les coûts de fonctionnement associés.

— Nous apportons plusieurs extensions à notre l'algorithme d'optimisation PigReuse afin d'améliorer sa performance, c'est-à-dire pour augmenter le nombre de sous-expressions communes qu'il peut être en mesure de détecter.

— Nous avons développé PigReuse comme un module d'extension du système Apache Pig. Nous apportons une évaluation expérimentale de nos techniques en utilisant deux fonctions de coût différentes pour sélectionner le plan le plus adapté.

Diffusion des connaissances. Le matériel présenté dans cette thèse en rapport avec ces travaux est proposé pour la publication dans une conférence internationale.

3. `COGROUP` peut être considéré comme une généralisation de l'opération *group-by* sur au moins deux relations : pour chaque valeur de clé de groupement de n'importe quelle entrée, il produira un tuple incluant un attribut *group* lié à la clé de groupement et un multi-ensemble de tuples pour chaque entrée R_i tel que le multi-ensemble R_i inclue tous les multi-ensembles de R_i contenant la valeur de la clé de groupement.

Conclusion

Le volume et l'hétérogénéité des données Web continuant d'augmenter rapidement, le besoin d'avoir des systèmes efficaces de gestion et d'extraction de l'information devient de plus en plus pressant.

Dans cette thèse nous avons exploré les performances et les coûts de l'entreposage de données Web dans des infrastructures commerciales cloud qui sont de nos jours faciles d'accès. De plus, nous avons exploré la parallélisation et l'optimisation des langages de requête pour le traitement des données sur ces infrastructures évolutives afin d'étudier le défi de la gestion des données auquel nous sommes aujourd'hui confrontés.

Plus précisément, nous nous sommes concentrés sur trois problèmes que nous rappelons ci-dessous.

Entreposage des données Web par l'utilisation de services commerciaux cloud. Nous avons présenté AMADA, une architecture pour l'entreposage des données Web via l'utilisation de services commerciaux cloud qui peut exploiter le parallélisme afin d'accélérer l'indexation et le traitement des requêtes. Au travers d'expériences, nous avons étudié et comparé différentes stratégies d'indexation pour les données XML et avons montré qu'elles permettent de réduire le temps d'exécution des requêtes et de diminuer les coûts financiers de plusieurs ordres de grandeur dans l'AWS.

Parallélisation de l'exécution de XQuery. Nous avons présenté PAXQuery, un processeur massivement parallèle des requêtes XML permettant de traduire XQuery en plans parallèles PACT. Nous avons ciblé un sous-ensemble riche de XQuery 3.0 incluant des ajouts récents tels que le groupement explicite et avons démontré à travers des expérimentations sur des ensembles de centaines de Go de données, l'efficacité et la scalabilité de PAXQuery.

Optimisation de la réutilisation pour le Pig Latin. Nous avons présenté une nouvelle approche permettant d'identifier et de réutiliser les sous-expressions qui surviennent dans les scripts Pig Latin. Plus précisément, nous avons posé les fondations de notre algorithme de réutilisation en formalisant la sémantique du langage de requête Pig Latin grâce au NRAB, étendue de façon à s'ajuster à la sémantique des opérateurs Pig Latin. Notre algorithme PigReuse identifie les possibilités de fusion de sous-expressions et sélectionne les meilleures à exécuter selon un processus de recherche basé sur les coûts mis en œuvre grâce à l'aide d'un solveur de programme linéaire. Nos résultats expérimentaux démontrent la valeur de nos algorithmes de réutilisation et de nos stratégies d'optimisation.

Références

[AABCR+12] Andrés Aranda-Andújar, Francesca Bugiotti, Jesús Camacho-Rodríguez, Dario Colazzo, François Goasdoué, Zoi Kaoudi, and Ioana Manolescu. AMADA : Web Data Repositories in the Amazon Cloud (demo). In *CIKM*, 2012.

[AWS] Amazon Web Services. http://aws.amazon.com/.

[BCM+13] Nicole Bidoit, Dario Colazzo, Noor Malla, Federico Ulliana, Maurizio Nolè, and Carlo Sartiani. Processing XML queries and updates on map/-reduce clusters (demo). In *EDBT*, 2013.

[BCRG+14] Francesca Bugiotti, Jesús Camacho-Rodríguez, François Goasdoué, Zoi Kaoudi, Ioana Manolescu, and Stamatis Zampetakis. SPARQL Query Processing in the Cloud. In Andreas Harth, Katja Hose, and Ralf Schenkel, editors, *Linked Data Management*, Emerging Directions in Database Systems and Applications. Chapman and Hall/CRC, April 2014.

[BEG+11] Kevin S. Beyer, Vuk Ercegovac, Rainer Gemulla, Andrey Balmin, Mohamed Y. Eltabakh, Carl-Christian Kanne, Fatma Özcan, and Eugene J. Shekita. Jaql : A Scripting Language for Large Scale Semistructured Data Analysis. *PVLDB*, 2011.

[BEH+10] Dominic Battré, Stephan Ewen, Fabian Hueske, Odej Kao, Volker Markl, and Daniel Warneke. Nephele/PACTs : a programming model and execution framework for web-scale analytical processing. In *SoCC*, 2010.

[BFG+08] Matthias Brantner, Daniela Florescu, David A. Graf, Donald Kossmann, and Tim Kraska. Building a database on S3. In *SIGMOD*, 2008.

[CLK+12] Hyebong Choi, Kyong-Ha Lee, Soo-Hyong Kim, Yoon-Joon Lee, and Bongki Moon. HadoopXML : A suite for parallel processing of massive XML data with multiple twig pattern queries (demo). In *ACM CIKM*, 2012.

[CRCM12] Jesús Camacho-Rodríguez, Dario Colazzo, and Ioana Manolescu. Building Large XML Stores in the Amazon Cloud. In *Data Management in the Cloud (DMC) Workshop*, 2012.

[CRCM13] Jesús Camacho-Rodríguez, Dario Colazzo, and Ioana Manolescu. Web Data Indexing in the Cloud : Efficiency and Cost Reductions. In *EDBT*, 2013.

[CRCM14] Jesús Camacho-Rodríguez, Dario Colazzo, and Ioana Manolescu. PAX-Query : A Massively Parallel XQuery Processor. In *DanaC '14*, 2014.

[DG04] Jeffrey Dean and Sanjay Ghemawat. MapReduce : Simplified Data Processing on Large Clusters. In *OSDI*, 2004.

[DPX04] Alin Deutsch, Yannis Papakonstantinou, and Yu Xu. The NEXT Logical Framework for XQuery. In *VLDB*, 2004.

[GCP] Google Cloud Platform. http://cloud.google.com/.

[GM93] Stéphane Grumbach and Tova Milo. Towards Tractable Algebras for Bags. In *PODS*, 1993.

[Had] Apache Hadoop. http://hadoop.apache.org/.

[KCS11] Shahan Khatchadourian, Mariano P. Consens, and Jérôme Siméon. Having a ChuQL at XML on the Cloud. In *AMW*, 2011.

[MHM06] Norman May, Sven Helmer, and Guido Moerkotte. Strategies for query unnesting in XML databases. *TODS*, 31(3), 2006.

[MPV09] Ioana Manolescu, Yannis Papakonstantinou, and Vasilis Vassalos. XML Tuple Algebra. In *Encyclopedia of Database Systems*. 2009.

[ORS+08] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig Latin : A Not-So-Foreign Language for Data Processing. In *SIGMOD*, 2008.

[Pig] Apache Pig. http://pig.apache.org/.

[SDQR10] Jörg Schad, Jens Dittrich, and Jorge-Arnulfo Quiané-Ruiz. Runtime Measurements in the Cloud : Observing, Analyzing, and Reducing Variance. *PVLDB*, 2010.

[Str] Stratosphere System. http ://stratosphere.eu/.

[TSJ+10] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Anthony, Hao Liu, and Raghotham Murthy. Hive - a PB scale data warehouse using Hadoop. In *ICDE*, 2010.

[WA] Windows Azure. http://www.windowsazure.com/.

[ZCD+12] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets : a fault-tolerant abstraction for in-memory cluster computing. In *NSDI*, 2012.