Abdelkader Hameurlain
Tran Khanh Dang
Franck Morvan (Eds.)

Data Management in Cloud, Grid and P2P Systems

LNCS 8648

7th International Conference, Globe 2014
Munich, Germany, September 2–3, 2014
Proceedings

Springer

Lecture Notes in Computer Science 8648

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Abdelkader Hameurlain Tran Khanh Dang
Franck Morvan (Eds.)

Data Management in Cloud, Grid and P2P Systems

7th International Conference, Globe 2014
Munich, Germany, September 2-3, 2014
Proceedings

Springer

Volume Editors

Abdelkader Hameurlain
Paul Sabatier University
IRIT
118, route de Narbonne
31062 Toulouse Cedex, France
E-mail: hameurlain@irit.fr

Tran Khanh Dang
HCMC University of Technology
268 Ly Thuong Kiet Street, District 10
Ho Chi Minh City, Vietnam
E-mail: khanh@cse.hcmut.edu.vn

Franck Morvan
Paul Sabatier University
IRIT
118, route de Narbonne
31062 Toulouse Cedex, France
E-mail: morvan@irit.fr

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-319-10066-1 e-ISBN 978-3-319-10067-8
DOI 10.1007/978-3-319-10067-8
Springer Cham Heidelberg New York Dordrecht London

Library of Congress Control Number: 2014945810

LNCS Sublibrary: SL 3 – Information Systems and Application, incl. Internet/Web and HCI

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

Data management involves a strategic stake for governmental organizations and companies, which need to anticipate and make relevant decisions based on high-quality decision support. In large-scale distributed enthronements, research activities in terms of data management continue intensively to explore new design approaches, processing methods, and implementation techniques for developing data management systems with the main characteristics such as scalability, elasticity, and self-managing.

The 7th International Conference on Data Management in Grid and P2P Systems (Globe 2014) was held during September 2–3, 2014 in Munich, Germany. The Globe Conference provides opportunities for academics and industry researchers to present, exchange, and discuss the latest data management research and applications in cloud, grid, and peer-to-peer systems.

Globe 2014 received 14 papers from 9 countries. The reviewing process led to the acceptance of 7 papers for presentation at the conference and inclusion in this LNCS volume. Each paper was reviewed by at least three Program Committee members. The selected papers focus on query processing and optimization, recommender systems, MapReduce framework (e.g., data privacy and similarity search), and data management in grid environments (e.g., protocol, recovery failure).

The conference would not have been possible without the support of the Program Committee members and members of the DEXA Conference organizing Committee and the authors. In particular, we would like to thank Gabriela Wagner and Roland Wagner (FAW, University of Linz) for their help in the realization of this conference.

June 2014

Abdelkader Hameurlain
Tran Khanh Dang
Franck Morvan

Organization

Conference Program Chairs

Abdelkader Hameurlain	IRIT, Paul Sabatier University, France
Tran Khanh Dang	HCMC University of Technology, Vietnam

Publicity Chair

Franck Morvan	IRIT, Paul Sabatier University, France

Program Committee

Fabricio B. Alves	FIOCRUZ Fundação Oswaldo Cruz, Brazil
Philippe Balbiani	IRIT, Paul Sabatier University, France
Nadia Bennani	LIRIS, INSA of Lyon, France
Djamal Benslimane	LIRIS, Universty of Lyon, France
Qiming Chen	HP Labs, USA
Thomas Cerqueus	University College Dublin, Ireland
Alfredo Cuzzocrea	ICAR-CNR, University of Calabria, Italy
Frédéric Cuppens	Telecom, France
Bruno Defude	Telecom INT, France
Mourad Elloumi	Tunis University, Tunisia
Tasos Gounaris	Aristotle University of Thessaloniki, Greece
Maria Indrawan-Santiago	Monash University, Australia
Sergio Ilarri	University of Zaragoza, Spain
Gildas Menier	LORIA, University of South Bretagne, France
Anirban Mondal	Xerox Research Lab, India
Riad Mokadem	IRIT, Paul Sabatier University, France
Franck Morvan	IRIT, Paul Sabatier University, France
Faïza Najjar	National Computer Science School, Tunisia
Kjetil Nørvåg	Norwegian University of Science and Technology, Norway
Jean-Marc Pierson	IRIT, Paul Sabatier University, France

Wenny Rahayu	La Trobe University, Australia
Claudia Roncancio	LIG, Grenoble University, France
Soror Sahri	LIPADE, Descartes Paris University, France
Florence Sedes	IRIT, Paul Sabatier University, France
Mário J. Gaspar da Silva	ST/INESC-ID, Portugal
Hala Skaf-Molli	LINA, Nantes University, France
David Taniar	Monash University, Australia
A Min Tjoa	IFS, Vienna University of Technology, Austria
Farouk Toumani	LIMOS, Blaise Pascal University, France
Roland Wagner	FAW, University of Linz, Austria
Wolfram Wöß	FAW, University of Linz, Austria
Jiangtao Yin	University of Massachusetts Amherst, USA
Shaoyi Yin	IRIT, Paul Sabatier University, France

Table of Contents

Optimizing Aggregate Query Processing in Cloud Data Warehouses

Swathi Kurunji, Tingjian Ge, Xinwen Fu,
Benyuan Liu, Amrith Kumar, and Cindy X. Chen

University of Massachusetts Lowell, MA, USA
{skurunji,ge,xinwenfu,bliu,cchen}@cs.uml.edu,
amrith@parelastic.com

Abstract. In this paper, we study and optimize the aggregate query processing in a highly distributed Cloud Data Warehouse, where each database stores a subset of relational data in a star-schema. Existing aggregate query processing algorithms focus on optimizing various query operations but give less importance to communication cost overhead (Two-phase algorithm). However, in cloud architectures, the communication cost overhead is an important factor in query processing. Thus, we consider communication overhead to improve the distributed query processing in such cloud data warehouses. We then design query-processing algorithms by analyzing aggregate operation and eliminating most of the sort and group-by operations with the help of integrity constraints and our proposed storage structures, PK-map and Tuple-index-map. Extensive experiments on PlanetLab cloud machines validate the effectiveness of our proposed framework in improving the response time, reducing node-to-node interdependency, minimizing communication overhead, and reducing database table access required for aggregate query.

Keywords: Aggregate Operation, Communication Cost, Read-Optimized Database, Data Warehouse, Cloud Storage, Query Optimization.

1 Introduction

Data Warehouses or decision support systems use join, group-by, and aggregate operations very often in formulating analytical queries. One of the survey conducted by Oracle [15] shows that, 36% of Data Warehouse users are having performance problems. Common performance bottlenecks include loading large data volumes into a data warehouse, poor metadata scalability, running reports that involve complex table joins and aggregation, increase in the complexity of data (dimensions), and presenting time-sensitive data to business managers etc.

Efficient evaluation of complex queries (i.e. aggregate and multi-join queries) is an important issue in applications that manage and analyze multidimensional data (analytical business data, scientific data, spatial data etc.). Efficient execution of such queries in large-scale and dynamic cloud databases is a challenging

A. Hameurlain et al. (Eds.): Globe 2014, LNCS 8648, pp. 1–12, 2014.

problem [7]. One of the main reasons is that we need to update global indexes (such as DHTs) every time the data is moved (or changed), or new machine is added. For example, in a ring network like Cassandra architecture, new machines are added to the ring near the bottlenecked machine, and the data is redistributed between that machine and its new neighbor.

One of the important properties of cloud architecture is elastic scalability. It must therefore support scale-out, where the responsibility of query processing (and the corresponding data) is distributed among multiple nodes to achieve higher throughput. Such network needs good storage structures, which reduce the dependency of data distribution on other machines of the cluster [3].

Our proposed PK-map and Tuple-index-maps are fully decentralized, where no predefined limits are imposed on the sizes of the network or data distribution. We only store information on relationship between the data and not their locations. It is independent of the scale-out of data or the remote data distribution. So, even if data is relocated remotely anywhere in the system, we do not have to update our structures. In our previous work [17], we have showed how our proposed method decreases the interdependency of machines (containing related data) while optimizing the join operation in the query processing. In this paper we will show how we optimize the aggregate query with join operations.

Aggregate operation is one of the expensive operations in distributed query processing due to the requirement of sort and group-by operations to find the aggregated result. Most of the earlier research work considers communication cost as cheaper of all other operations of the query.[1] Hence two-way optimization algorithm is most commonly used, where query optimizer first generates the plan assuming that the query is processed in local machine and, then optimizes the plan considering the distributed architecture of query execution.

Due to virtualization nature of a cloud environment, data storage and query processing have become physically more distributed to meet the resource availability or the customers service agreement [9]. This gives rise to increase in node-to-node communication. In such scenario, even if the overall response time of distributed query is decreased, it is possible that communication cost exceeds other query operator costs. So, it is better to give more consideration to both query operators as well as communication cost while generating the plan.

In this paper, we use our map structures and integrity constraint inference to push down or pull up group-by functions while generating an aggregate query plan. During query processing, we eliminate most of the sort and group-by operations by having sorted map structures and enforcing a certain order based on the hierarchy of the tables in the schema (reference graph of [17]). We will show how we only access more relevant data from the database tables.

The remainder of this paper is organized as follows: Section 2 provides a literature review on aggregate query processing and optimization. Section 3 explains our proposed framework. Section 4 shows the performance evaluation using PlanetLab Cloud. Finally, Section 5 states the conclusion.

[1] Communication cost includes costs per message, costs to transfer data and CPU costs to pack, unpack, and process messages at the sending and receiving sites.

2 Related Work

Aggregate query processing has been studied in many research works [5]. But, as per our knowledge, not many of them consider communication cost in optimizing aggregate query processing. We analyzed some of the works which optimize the aggregate query operations. Along with that knowledge, we propose our storage structures, which will not only optimize query operations, but also communication cost overhead caused in cloud data warehouses.

Some of the earlier papers, which optimize aggregate query processing, are [2] [14] and [22]. These papers provide optimizations by pushing down group-by in the query tree to improve the query response time. W.Yan [22] proposed two kinds of transformations namely, eager aggregation and lazy aggregation. In eager aggregation, group-by operation is pushed down in the query tree, while in lazy aggregation group-by is pushed up. We use the above transformations of [22] in our system along with our PK-map and Tuple-index-map to generate optimized query plan to process aggregate queries.

Order-Optimization [4], presents techniques to reduce the number of sorts needed for query processing by finding the cover set using keys, predicates and indexes. Since our proposed map structures are already sorted on keys, we eliminate most of the sort operations required for join operation on the tables.

Coloring-Away [23], proposed query plan generation using tree-coloring mechanism. This paper considers both communication cost and data re-partitioning, and uses tree coloring to generate optimal query plan. In our framework, we optimize the query operations that cause the above mentioned query performance problems such as aggregates and joins by doing sort and group-by on the fly.

Avoid-Sort-Groupby [24], proposed a query plan refining algorithm through which unnecessary sorting and grouping can be eliminated from the query plan. It uses inference strategies and order properties of the relation table to find the unnecessary sorting or grouping. T.Neumann[19], points out that it is necessary to consider both ordering and grouping to generate the query plan.

Cooperative-Sort [25], presented an evaluation technique for sorting tables. This technique is for those queries that need multiple sort orders of the same table on different attributes. This minimizes the I/O operations of successive sort operations, which reduce the overall query cost.

Pre-computing the aggregates is proposed by many other researchers [6] [16], which are useful for decision support systems. Decision support systems store huge amount of historical data for analysis and decision-making. These databases are updated less frequently (once a hour/day) on batches. This made it easy to compute the aggregation ahead of time and store it as data cubes or materialized views. Recently, the interval between historic and current data has been reduced a lot. This will make it complicated and time consuming to re-compute data cubes or materialized views every time data gets updated. Recent research by companies like HP, Oracle and Teradata [8] [18] [21] shows new parallelization schemes for processing joins and aggregate operations, eliminating data cubes. So, in this paper we concentrate on optimizing aggregate queries without pre-computation.

3 Our Approach

During query execution, a query accesses data from different tables by performing join operation on primary and foreign key attributes of those tables, and then the required attribute values are filtered. This becomes more complicated in the virtual cloud environment, where the location of the data changes more often based on the resource availability or SLA (Service Level Agreement) of the customer. Increased node-to-node communication results in increased network delay and processing time, leading to the decrease in query performance. Hence, we need storage structures that help to minimize the communication between machines, minimize the maintenance during updates and minimize the increase in the storage space.

3.1 Storage Structure

PK-map Structure: We create a PK-map (i.e., Primary Key map) for each of the primary keys in the tables of the star schema. A PK-map will have one column for the primary key and one column for each of the foreign keys referencing this primary key as shown in Table 1. Primary key column contains primary key values and foreign key column contains logical record-ids. These logical record-ids are index to the Tuple-index-map of the foreign key. Tuple-index-map is explained in the next subsection. The PK-maps are sorted on primary key values, which allow us to apply run length encoding on foreign key logical record-ids and reduce the size of the map to a great extent. Thus, the overall size of the map will be proportional to the number of records in the dimension table, which are usually smaller in size than fact table.

Tuple-index-map Structure: We create a Tuple-index-map for each of the foreign key attribute, which reference the primary key attribute of the dimension table as shown in Table 2. This Tuple-index-map will store the mapping between the logical and actual record-id of the foreign keys in the foreign key table.

Table 1. NationKey-map and SuppKey-map

	$N.nationkey$	$S.nk\_index$	$C.nk\_index$
Row 0	0000000000	0	0
Row 1	0000000001	4	6
Row 2	0000000002	14	26
..	.....	.....	.....
..	.....	.....	.....
Row 24	0000000024	.....	.....

	$S.suppkey$	$PS.sk\_index$	
Row 0	0000000000	0	
.....	.....	.....	Partition 1
Row 20000	0000020000	100000	
Row 20001	0000020001	.....	Partition 2
.....	.....	.....	
.....	.....	.....	.....
.....	.....	.....	Partition 49
Row 980001	0000980001	.....	
.....	.....	.....	Partition 50
Row 1000000	0001000000	.....	

Table 2. SuppKey Tuple-index-map

$PS.sk\_index$	$PS.sk\_t\_index$	
Row 0	0	
.....	24	Partition 1
.....	.....	
Row 400000	.....	
Row 400001	.....	Partition 2
.....	.....	.....
.....	.....	.....
.....	.....	.....
.....	.....	.....
.....	.....	.....
.....	.....	.....
.....	.....	Partition 49
Row 19600000	.....	
.....	.....	
.....	.....	Partition 50
.....	.....	
Row 20000000	.....	

If the foreign key table is sorted on the foreign key attribute value, then we do not require Tuple-index-map for that relationship. In that case, logical record-id of the PK-map will be the actual record-id of the foreign key table. For example, in Figure 1, PART and PARTSUPP tables are sorted on partkey attribute. Hence, PartKey-map does not require Tuple-index-map. Similarly, in column-oriented databases, if at least one projection of the table is sorted on foreign key attribute, then the logical record-id of the PK-map will be the actual foreign key record-id.

3.2 TPC-H Star Schema

Star and Snowflake Schema representations are commonly used in read-optimized Data Warehouses. In the rest of this paper we use the star schema from TPC BENCHMARK H Standard Specification Revision 2.15.0 (Figure 1) for analysis and performance study. Figure 1 is the schema of an industry, which must manage, sell and distribute its products worldwide.

With the TPC-H schema of Figure 1, we need to create 6 PK-maps NationKey-map, SuppKey-map, PartKey-map, PartsuppKey-map, CustKey-map and OrderKeymap. Structure of NationKey-map and SuppKey-map are shown in Table 1. We also need 5 Tuple-index-maps (like Table 2) for relationships between each of the following tables: nation $\leftrightarrow$ supplier, nation $\leftrightarrow$ customer, supplier $\leftrightarrow$ partsupp, customer $\leftrightarrow$ orders table and partsupp $\leftrightarrow$ supplier.

Through performance study of our previous work [17], we know that the size of the maps would usually be around 10% to 12% of the actual data size. Size can be calculated using below formulas.

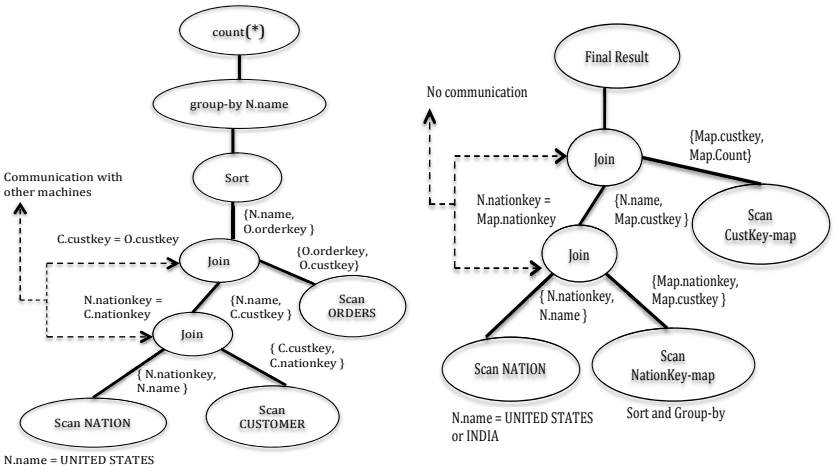
$$Size\ of\ PK\ map = S_1 + \sum_{i=0}^{n} S_2[i] + c \tag{1}$$

$$Size\ of\ Tuple\_index\_map = (Number\ of\ rows\ in\ FK\ Table)\ *\ S_2 \tag{2}$$

where, S_1 is the size of the primary key and S_2 is the size of logical record id (32/64 bit depending on the type of processor)

REGION(regionkey, name, comment)
NATION(nationkey, name, regionkey, comment)
SUPPLIER(suppkey, name, address, nationkey, phone, acctbal, comment)
CUSTOMER(custkey, name, address, nationkey, phone, acctbal, mktsegment, comment)
PART(partkey, name, mfgr, brand, type, size, container, retailprice, comment)
PARTSUPP(partkey, suppkey, availqty, supplycost, comment)
ORDERS(orderkey, custkey, orderstatus, totalprice, orderdate, order-priority, clerk, ship-priority, comment)
LINEITEM(orderkey, partkey, suppkey, linenumber, quantity, extendedprice, discount, tax, returnflag, linestatus, shipdate, commitdate, receiptdate, shipinstruct, shipmode, comment)

Fig. 1. TPC-H Benchmark Schema [20]



(a) MySQL Plan (b) PK-map Plan

Fig. 2. Query Processing Plan for Query1 of Table 3

3.3 Aggregate Query Processing

Inferring on referential integrity constraints and functional dependencies, we have classified aggregate operations into two general categories based on the type of attribute on which the aggregate operation is applied. We then generate plan for query processing accordingly.[2]

Aggregate on Primary or Foreign Key: When there is an aggregate operation on primary key (PK) or foreign key (FK), we do not scan the database table (unless there is a filtering constraint on non-PK or non-FK), instead we scan our proposed map structures and perform aggregate operation on the fly. We can do this because, our maps are sorted on keys, and has required information on number of tuples of foreign key table that are mapped to primary key of dimension table.

For example, Query 1 in Table 3 has count(*) operation. This query is trying to find the total number of orders placed by the specified nations. Here, we only require the number of rows in orders table that belong to each group in the group-by clause (i.e., N.name). By inference, we can say that, this is an aggregate operation on PK of the orders table. Hence, we can eliminate the scan of orders table and get this count by scanning CustKey-map. We push down group-by on N.name to step "scan NationKey-map" as shown in Figure 2b.

Figure 2a is the plan generated by MySQL for processing Query 1 of Table 3. As shown in Figure 2a, it requires three tables scan (Nation, Customer and Orders), two joins, a sort and a group by operation. Here, query processor needs to communicate the data for each join operation, which increases the communication overhead and response time.

[2] We do not consider "having" clause in our analysis, because having clause can be converted into "where" clause constraints by rewriting the query [22].

```
select N.name, SUM(O.totalprice)
from ORDERS O, CUSTOMER C, NATION N
where N.name = "UNITED STATES"
        or N.name = "INDIA"
        and N.nationkey = C.nationkey
        and C.custkey = O.custkey
group by N.name;
```

Fig. 3. Modified Query 1 of Table 3

On the other hand, Figure 2b is the plan generated for the same Query 1 using our framework. Here, instead of scanning Customer and Orders table, we scan two PK-maps (NationKey-map and CustKey-map), and its associated Tuple-index-maps. These maps are very small compared to scanning tables. The join operation in Figure 2b is not equivalent to join operation in Figure 2a. Instead, it is performed by scanning map and applying associated filtering constraint. Algorithms are shown in Figure 4a - 4b and comparison results in Section 4.

Aggregate on Non-Primary or Non-Foreign Key: When there is an aggregate operation on Non-Primary Key (NPK) or Non-Foreign Key (NFK) of the table, then we need to scan the table to find the correct result. But, we use inference to push down the group-by in the query tree so that we can scan only the required portion of table. We also replace the scanning table with maps wherever possible.

For example, suppose if we have SUM (O.totalprice) instead of COUNT (*) of Query 1 as shown in Figure 3. We push down the group-by N.name (as in Figure 2b) to reduce the number of rows to be scanned from the orders table. This reduces the input rows of final aggregate operation.

Group-by Optimization: When there are multiple attributes in the group-by clause, we associate certain order to those attributes and push down group-by in the query tree. As we know that the group-by is a set operation and join result will carry the sort order, we can change the sequence of attributes in group-by clause. So, we change the sequence of attributes in group-by clause corresponding to the sequence of table processing in our algorithm. By doing this we can eliminate non-relevant data in the early stage of query processing as well as achieve same result as we perform group-by in the final stage.

Query Processing Using Proposed Method: In Algorithm 1 shown in Figure 4, we first sort the tables referenced in the query according to their relationship in the schema. We sort tables starting from the table that does not have any foreign keys (depth 0) [17]. For example, Query 1's order of processing will be nation $\rightarrow$ customer $\rightarrow$ orders. In addition, we consider all the constraints in *where* clause and *group-by* while deciding on tables processing order.

We then consider aggregate operation to decide whether to scan the table from the database or to scan our map structure (Line 3-15). If there are PK or FK constraints, we scan our maps. As all PK to FK mapping information is available in maps, data movement between nodes is eliminated in lines 5-8.

Algorithm 1 Aggregate Query Processing Algorithm

Input: query Q, reference graph G [17]
Output: Result of query Q
1: Let T be an array of tables referenced in Query
2: **Sort** T based on *d* value of G and filtering constraints
//d value is shown in reference graph of [17]
3: **for each** table *t ε T* **do**
4: Perform predicate/join processing using Algorithm 2
5: **if** there is an aggregate on PK or FK **then**
6: Scan PK-map
7: //sort and group-by is inferred
8: Update the current result set
9: **else if** there is an aggregate on NPK or NFK **then**
10: Scan t and apply aggregate operation
11: //sort and group-by is applied if necessary
12: Update the current result set
13: Communicate if necessary with other machines
14: **end if**
15: **end for**
16: Scan tables of T if necessary to get the value of other attributes in the select statement of Q
17: **return** Result

(a) Aggregate Query Processing Algorithm

Algorithm 2 Predicate/Join Processing Algorithm

Input: Table t ∈ *T*, predicates of Q, required attributes in result
Output: Result of t
1: **if** there is a predicate on non-PK/non-FK **then**
2: **if** d == 0 for t **then**
3: Apply predicate on t to get the record ids
4: Store the record-id mapping in the format
5: (rec-id_1, rec-id_2,….)
6: Communicate if necessary with other nodes
7: **else if** any table t_1 with d_1 <= d referenced by t **then**
8: Apply predicate on t
9: Update the mapping with rec-ids of t
10: Perform line 9
11: Eliminate mappings which has no match for t
12: **else**
13: Perform similar to line 6, 9 and 14
14: **end if**
15: **else if** there is a predicate on PK or FK **then**
16: **if** d == 0 for t **then**
17: Scan PK-map and tuple-index-map
18: Perform line 6 to 8
19: **else**
20: Scan PK-map and tuple-index-map for those rec-ids stored for table t_1 with d_1 <= d that is referenced by t
21: Perform 12 and 14
22: **end if**
23: **end if**
24: Scan tables of T for final mappings (rec-id_1,…….) to get the value of other attributes in the select statement of Q
25: **return** Result

(b) Join Processing Algorithm [17]

Fig. 4. Query Processing Algorithms

Also, our maps are already sorted on keys which further eliminates most of the sort operations. At last we retrieve remaining attributes required for the result.

4 Performance Evaluation

In this section, we present the performance study to show the effectiveness of our proposed PK-map and Tuple-index-map structures while processing aggregate queries (using Algorithms in Figure 4a and Figure 4b). We will compare the performance between MySQL and our proposed framework on a large-scale cloud network called PlanetLab with 150GB of TPC-H star schema data.

PlanetLab [12] [13] is a geographically distributed computing platform available as a testbed for deploying, evaluating, and accessing planetary-scale network services. It is currently composed of around 1050 nodes (servers) at 400 sites (location) worldwide.

For performance study of this paper we chose 50 PlanetLab machines worldwide running Red Hat 4.1 Operating System. Each machine has 2.33GHz Intel Core 2 Duo processor, 4GB RAM and 10GB disk space. We installed regular MySQL on all of the machines to perform experiments.

We generated 150GB of data using the data generator "dbgen", provided by TPC-H benchmark and distributed it to 50 PlanetLab machines. Each of these machines store around 3GB data fragments of TPC-H schema relations. We generated PK-maps and Tuple-index-maps, and then horizontally partitioned them

using the same partition key that was used to partition the data. We then distributed these partitions into all 50 machines that contained corresponding data. All the map structures are loaded into the main memory before each experiment begins. Here we can take advantage of main memory databases if the data size is huge (in Petabyte or Exabyte) [1] [10] [11].

We used 5 queries for the performance analysis as shown in Table 3. Comparison of time taken by these queries is shown in Figure 5a to 5e. In graphs of these figures, PlanetLab machines are on the x-axis and time taken by the query (in seconds) is on the y-axis.

On each graph, top line shows the result of processing aggregate queries on MySQL, and the bottom line shows the result of our framework. For simplicity of explanation, we do not consider pipelined approach to count the number of communications between the machines. Instead we use inter query operation communications.

Figure 5a compares the time taken for processing Query 1. As explained in Section 3.3.1, Query 1 has an aggregate operation on primary key of table ORDERS, and a Non-Primary Key/Non-Foreign Key (NPK/NFK) constraint on table NATION. Hence, it is sufficient to scan NATION table, NationKey-map, and CustomerKey-map to answer this query.

Figure 5b compares the time taken for processing Query 2. In Query 2, we have an aggregate operation COUNT(*), and a NPK constraint on attribute

Table 3. Performance Evaluation Queries

Query 1*: (COUNT) Find the total number of orders placed by the 'UNITED STATES' and 'INDIA' customers (aggregation on primary key attribute)* select N.name, count(*) from ORDERS O, CUSTOMER C, NATION N where N.name = "USA" or N.name = "INDIA" and N.nationkey = C.nationkey and C.custkey = O.custkey group by N.name;
Query2*: (COUNT) Find the total number of failed orders placed by customers of each nation. I.e, orders whose ORDERSTATUS = 'F' (aggregation on NON - Primary Key attribute)* select N.name, count(*) from ORDERS O, CUSTOMER C, NATION N where N.nationkey = C.nationkey and C.custkey = O.custkey and O.orderstatus = 'F' group by N.name;
Query 3*: (MIN) Find the minimum supply cost for all the parts supplied by 'UNITED STATES' suppliers* select PS.partkey, min(PS.supplycost) from PARTSUPP PS, SUPPLIER S, NATION N where N.name = "USA" and N.nationkey = S.nationkey and S.suppkey = PS.suppkey group by PS.partkey;
Query 4*: (SUM) Find the revenue generated by customers of each nation in the year 1995. I.e., revenue is equal to the totalprice from the orders table* select N.name, sum(O.totalprice) from ORDERS O, CUSTOMER C, NATION N where N.nationkey = C.nationkey and C.custkey = O.custkey and O.orderdate like '1995%' group by N.name;
Query5*: (AVG) Find the average revenue generated by customers of each nation in year 1995. I.e., revenue is equal to the totalprice from the orders table* select N.name, avg(O.totalprice) from ORDERS O, CUSTOMER C, NATION N where N.nationkey = C.nationkey and C.custkey = O.custkey and O.orderdate like '1995%' group by N.name;

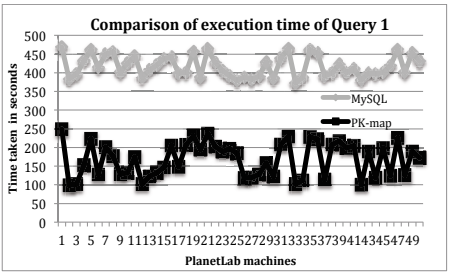
O.orderstatus of table ORDERS. Hence, we need to scan ORDERS table along with NationKey-map and CustomerKey-map to process this query.

Figure 5c compares the time taken for processing Query 3. To answer this query, we need to scan PARTSUPP table, NATION table, NationKey-map, and SuppKey-map.

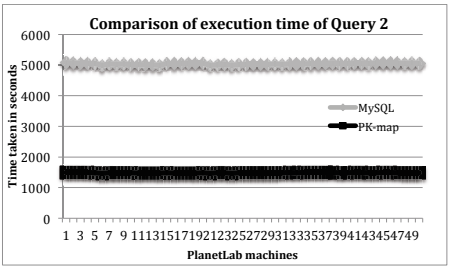
Figure 5d compares the time taken for processing Query 4. This query has an aggregate operation and a filtering constraint on NPK of table ORDERS. Hence, we scan NationKey-map, CustKey-map, and ORDERS table to process the query.

Figure 5e compares the time taken for processing Query 5. This query is similar to Query 4, but needs to keep track of "count" in order to find the "AVG".

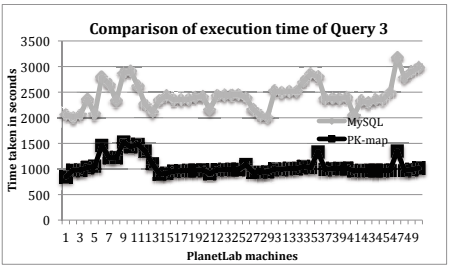
In the above experiments, MySQL takes 3 inter-machine communications, while our framework takes only one communication. In MySQL, each node has



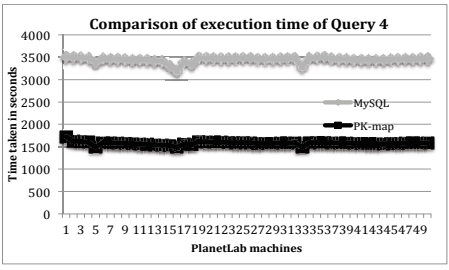
(a) Query 1



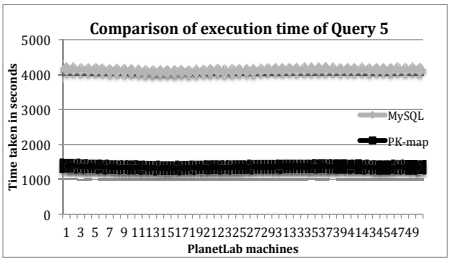
(b) Query 2



(c) Query 3



(d) Query 4



(e) Query 5

Fig. 5. Performance Comparison of Query 1 to Query 5 of Table 3

to communicate partial result or the join attribute values with its peers for every join predicate present in the query. Thus each node has to do additional message processing. This will effect the overall time taken by the query. But, with our map structures we do not communicate for every join operation reducing the communication delay. This less interaction between the database machines in the cluster reduces the communication overhead and effect of poor load balancing. Based on the data in Figure 5a to 5e, it is clear that, the query execution time required by the state-of-the-art MySQL is more than our framework.

5 Conclusion

In this paper, we studied distributed aggregate query processing on cloud data warehouses. We proposed storage structures PK-map and tuple-index-map, and designed query-processing algorithm for processing these aggregate queries. We analyzed the query processing, and advantages of using maps in eliminating sort and group-by operations required by the aggregate operation. Our framework not only reduces the communication cost but also makes minimum access to relation tables depending on the type of aggregate operation. Results of our extensive performance study demonstrate that the proposed approach improves the performance of the ad-hoc aggregate query in Cloud Data Warehouses and reduces communication overhead.

Acknowledgments. This work is supported in part by the NSF, under the grants IIS-1149417, IIS-1239176, and IIS-1319600.

References

1. Kemper, A., Neumann, T.: Hyper: A hybrid OLTP and OLAP main memory database system based on virtual memory snapshots. In: IEEE 27th International Conference on Data Engineering (ICDE), Hannover, Germany, pp. 195–206 (2011)
2. Vlachou, A., Doulkeridis, C., Norvag, K., Kotidis, Y.: Peer-to-Peer Query Processing over Multidimensional Data. Springer (2012)
3. Curino, C., Evan, P.C.J., Raluca, A.P., Malviya, N., Wu, E., Madden, S., Balakrishnan, H., Zeldovich, N.: Relational Cloud: A Database Service for the cloud. In: 5th Biennial Conference on Innovative Data Systems Research (CIDR), California, USA, pp. 235–240 (2011)
4. Simmen, D., Shekita, E., Malkemus, T.: Fundamental Techniques for Order Optimization. In: ACM SIGMOD International Conference on Management of Data, Montreal, Canada, vol. 25, pp. 57–67 (1996)
5. Kossmann, D.: The state of the art in distributed query processing. ACM Computing Surveys (CSUR) 32(4), 422–469 (2000)
6. Xin, D., Han, J., Li, X., Benjamin, W.W.: Star-Cubing: Computing Iceberg Cubes by Top-Down and Bottom-Up Integration. In: 29th International Conference on Very Large Data Bases (VLDB), Berlin, Germany, vol. 29, pp. 476–487 (2003)
7. Boukhelef, D., Kitagawa, H.: Efficient Management of Multidimensional Data in Structured Peer-to-Peer Overlays. In: 35th International Conference on Very Large Data Bases (VLDB), vol. 35. Lyon, France (2009)

8. Graefe, G.: New algorithms for join and grouping operations. Journal Computer Science - Research and Development 27(1), 3–27 (2012)
9. Soundararajan, G., Lupei, D., Ghanbari, S., Adrian, D.P., Chen, J., Amza, C.: Dynamic Resource Allocation for Database Servers Running on Virtual Storage. In: 7th USENIX Conference on File and Storage Technologies (FAST), San Francisco, California, USA, pp. 71–84 (2009)
10. Garcia-Molina, H., Salem, K.: Main Memory Database Systems. IEEE Transactions on Knowledge and Data Engineering (TKDE) 4(6), 509–516 (1992)
11. Plattner, H.: A common database approach for OLTP and OLAP using an in-memory column database. In: ACM SIGMOD International Conference on Management of Data, Providence, USA, pp. 1–2 (2009)
12. Peterson, L., Roscoe, T.: The design principles of PlanetLab. In: ACM SIGOPS Operating Systems Review, New York, USA, pp. 11–16 (2006)
13. Planetlab Cloud, http://www.planet-lab.org/
14. Chaudhuri, S., Shim, K.: Including Group-By in Query Optimization. In: 20th International Conference on Very Large Data Bases (VLDB), Santiago, Chile, pp. 354–366 (1994)
15. Survey, http://www.oracle.com/us/products/database/high-performance-data-warehousing-1869944.pdf
16. Agarwal, S., Agrawal, R., Prasad, M.D., Gupta, A., Jeffrey, F.N., Ramakrishnan, R., Sarawagi, S.: On the Computation of Multidimentional Aggregates. In: 22nd International Conference on Very Large Data Bases (VLDB), Mumbai (Bombay), India, vol. 22, pp. 506–521 (1996)
17. Kurunji, S., Ge, T., Liu, B., Chen, C.X.: Communication Cost Optimization for Cloud Data Warehouse Queries. In: 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings (CloudCom), Taipei, Taiwan, pp. 512–519 (2012)
18. Srikanth, B., Li, H., Unmesh, J., Zhu, Y., Vince, L., Thierry, C.: Adaptive and Big Data Scale Parallel Execution in Oracle. International Journal on Very Large Data Bases (VLDB) 6(11), 1102–1113 (2013)
19. Neumann, T., Moerkotte, G.: A Combined Framework for Grouping and Order Optimization. In: 30th International Conference on Very Large Data Bases (VLDB), Toronto, Canada, vol. 30, pp. 960–971 (2004)
20. TPC-H benchmark, http://www.tpc.org/tpch/spec/tpch2.14.4.pdf
21. Teradata, http://www.teradata.com/white-papers/Teradata-Aggregate-Designer-eb6110
22. Weipeng, P.Y., Per-Ake, L.: Eager Aggregation and Lazy Aggregation. In: 21st International Conference on Very Large Data Bases (VLDB), Zurich, Switzerland, pp. 345–357 (1995)
23. Hasan, W., Motwani, R.: Coloring Away Communication in Parallel Query Optimization. In: 21st International Conference on Very Large Data Bases (VLDB), Zurich, Switzerland, pp. 239–250 (1995)
24. Wang, X., Cherniack, M.: Avoiding Sorting and Grouping in Processing Queries. In: 29th International Conference on Very Large Data Bases (VLDB), Berlin, Germany, vol. 29, pp. 826–837 (2003)
25. Cao, Y., Bramandia, R., Chan, C.-Y., Tan, K.-L.: Sort-Sharing-Aware Query Processing. International Journal on Very Large Data Bases (VLDB) 21(3), 411–436 (2012)

A General Nash Equilibrium Semantic Cache Algorithm in a Sensor Grid Database

Qingfeng Fan and Karine Zeitouni

University of Versailles-Saint-Quentin Laboratory PRISM,
78035 Versailles Cedex, France
qingfeng.fan@prism.uvsq.fr

Abstract. Sensor grid databases are powerful, distributed, self-organizing systems that allow in-network query processing and offer a user friendly SQL-like application development. We propose an adaptation of a well-known cache-based optimization and cache replacement policy to this context. Since the data are distributed and the sensor nodes are mobile, the cost model is more complicated than in traditional query optimization, because it should account for several factors, including the semantics, location and time. Therefore, we need a trade-off between those constraints. Our approach is based on a theoretical foundation for the game and balance problem. In summary, we propose an approach that (i) Based on the Nash Equilibrium scheme, an application of three scalar coefficients is proposed in term of the analysis of the relationship among semantic, time and location factors. (ii) After that, we specialize and summarize the general Nash Equilibrium scheme utilizing the equal correlation coefficient among the new and old vectors to find the point of Nash equilibrium and resolves the scalar coefficients. (iii) It uses these scalar coefficient to attain an optimum cost model of the semantic cache, for query optimization in the context of distributed query processing. (iv) We emphasize that this method can extend to any finite-dimension, which are other schemes cannot do. Extended simulation results indicate that our scheme outperforms existing approaches in terms of both the response time and the cache hit ratio.

Keywords: Sensor Grid Database, Semantic Cache, Location Dependent Query, Query Optimization, Nash Equilibrium.

1 Introduction

A Sensor Grid is a grid that gathers, distributes, and acts on information about the behavior of all participants including suppliers and consumers. In mobile sensor applications, users who carry portable devices such as mobile phones can issue local queries i.e, queries that are dependent on the user's location, to learn about their geographic surroundings [6]. However, those query operations are usually frequent, whereas the resources of the sensor grid are limited. Therefore, an optimal approach to handle local queries is needed. However, those Location Dependent Queries become expensive in this environment, due to the sensor nodes's resource limitations and the mobile communications cost. Hence, we

A. Hameurlain et al. (Eds.): Globe 2014, LNCS 8648, pp. 13–24, 2014.

investigate query optimization in the context of distributed query processing in a Sensor Grid Database.

Caching is a promising technique, widely used for query optimization. However, as the query cost model in sensor grids depends on different parameters than in the centralized setting, the cache management and the cache replacement policy needs to be adapted. In this paper, we propose two techniques: First, we propose a semantic cache, which is an optimized data structure for improving the query efficiency. Generally speaking, semantic cache stores the query result and query semantic in the cache, for responding the future query; Second, we propose a cache replacement policy, accounts for the location dependent queries cost model, keeping in the cache the most relevant data for better efficiency [1].

However we find that traditional approaches have a limitation. For example, the paper in [4] proposes the approach of using a Clustering Semantic Cache (CSC), which considers semantic and time factors, however there is only qualitative analysis and no quantitative description. In a previous paper [5], the cost model of the branch and bound and Greedy Dual-Size Frequency ($B\&B_{GDSF}$) considers semantic and time factors, which provides the advantage of quantitative analysis, but , since the geographical context where rather static, the model did not consider any dynamic location factor. In another paper [2], the cost model of Collaborative Spatial Data Sharing ($CSDS$) considered a dynamic location factor, but did not consider semantic and time factors. An advanced application needs the new approach of semantic cache technology to synthetically consider the multiple factors that are present.

How to organize the data in a mobile sensor device for access and query; how to calculate the cost of a cache item for replacement and storage; and how to synthetically consider semantic, time and location factors, and attain an optimum cost model of a semantic cache are very challenging issues.

Our contribution is summarized as follows. We propose an approach that has the following characteristics: (i) Based on the Nash Equilibrium scheme, an application of three scalar coefficients is proposed in term of the analysis of the relationship among semantic, time and location factors. (ii) After that, we specialize and summarize the general Nash Equilibrium scheme utilizing the equal correlation coefficient among the new and old vectors to find the point of Nash equilibrium and resolves the scalar coefficients. (iii) It uses these scalar coefficient to attain an optimum cost model of the semantic cache, for query optimization in the context of distributed query processing. (iv) We emphasize that this method can extend to any finite-dimension, which are other schemes cannot do.

2 Related Work

We choose three different ways for choosing and calculating a cost model for the replacement of semantic cache segments.

1) Clustering Based Semantic Cache CSC: The paper [4] proposes the approach of a Clustering Semantic Cache (CSC), which considers semantic and time factors but only performs qualitative analysis, with no quantitative description. Early in the algorithm the technique of Clustering Semantic Cache divides

queries into groups, and those that are semantically related queries are placed together in a cluster.

2) The Semantic Cache Based on $B\&B_{GDSF}$: In a previous paper [5], the cost model of the branch and bound and Greedy Dual-Size Frequency ($B\&B_{GDSF}$) considers semantic and time factors, moving from qualitative to quantitative analysis, but does not consider a dynamic location factor. When a new query result must be stored in a saturated cache, the most irrelevant queries must be evicted. It is important to account for not only the constraints on the size but also the cost and frequency of access to spatial objects. Thus, we base our replacement policy on the ($B\&B_{GDSF}$) algorithm proposed by Savary et al. This strategy replaces the object with the smallest key value for a certain cost function. When an object i is requested, it is given a priority key K_i that is computed as follows:

$$K_i = F_i * \frac{C_i}{S_i} + L. \tag{1}$$

where F_i is the frequency of usage of the object I, and includes not only how often node i uses the data, but also how often the data is requested from its cache by other nodes; C_i is the cost associated with bringing the object i into the cache, and C_i is a function of $C_{CacheCPU}$ and $C_{DiskCPU}$: $C_i = f(C_{CacheCPU}, C_{DiskCPU})$: $C_{CacheCPU}$ stands for the cost in CPUs of the cached query result, being the former the amount of processing required to extract something from the cache; and $C_{DiskCPU}$ stands for the cost in CPUs to re-compute the query result I from the database, being the latter the amount of processing required to extract something from a server. But for simple and effective discussion, in the simulation of our paper C_i is a constant; S_i is the size of the object I; and L is a running life factor that starts at i and is updated for each replaced object O to the priority key of this object in the priority queue, i.e., $L = K_0$.

The cache replacement policy is composed of two steps. First, the cost of each query result contained in cache is computed using the $B\&B_{GDSF}$ policy. Second, the total size of the set must be equal to or greater than the size of the new query result.

3) The Semantic Cache Based on of $CSDS$: In another paper [2], the cost model of Collaborative Spatial Data Sharing ($CSDS$) considers a dynamic location factor but does not consider semantic and time factors.

For Nodes A and B, $Prio_A$ and $Prio_B$ are the values of the replacement priority. For example, in the scenario in Fig. 1, Node A stays in a place that is at a distance of 10 meters for approximately 5 minutes and in the a place at a distance of 5 meters for approximately 5 minutes, which causes $Prio_A = 10meters \times 5minutes + 5meters \times 5minutes = 75$; Node B stays in a place at a distance 10 for approximately 10 minutes. $Prio_B = 10meters \times 10minutes = 100$, thus $Prio_B > Prio_A$, and the replacement priority of B is higher.

We use $Prob(M_i, E_j, \Delta t)$ to denote the probability that a device M_i will access the cell E_j during the time period $[t_c, t_c + \Delta t]$, where t_c is the current time. The cell E_j is a cell of a grid defined over a geographical area, where the mobile devices operate and wish to extract location-dependent data. We thus look Δt time units into the future and predict how likely M_i will need E_j at that time.

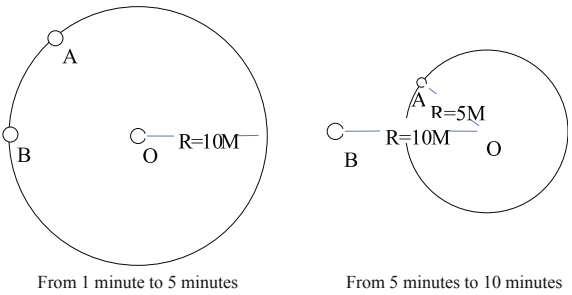


Fig. 1. The Integral of the Distance Over Time

$$Prob(M_i, E_j, \Delta t) \approx iDist_M(M_i, E_j, \Delta t)$$
$$= \int_{t_c}^{t_c+\Delta t} dist_M(M_i.pos(t), E_j)dt = \frac{1}{2}.|V_k|.((t_x - t_c)^2 + (t_f - t_x)^2).$$

3 Synthetically Considering Semantic, Time and Location Factors

The traditional semantic cache approaches, have a limitation in that they cannot satisfy the development of modern sensor grid technology. An advanced application needs the new approach of semantic cache technology to synthetically consider multiple factors. The main difference between semantic cache management and a conventional semantic cache is the cost model, which should account for several factors including semantic, location and time. Therefore, we need a trade-off between those constraints [3].

Our approach is the comprehensive consideration of the semantic, time and location balance of the 3-Dimensional vectors Nash Equilibrium. There is much research on special algorithms for any special factors of $3 - D$ factors. Our approach has been developed for a Nash Equilibrium scheme with 3-Dimensional vectors. Moreover we can generally and practically propose to any finite M-Dimension vectors algorithm. Our algorithm is organically combined with these 3-D factors, and can be easily expanded.

3.1 To Define the Weights of the Nodes

$$W_{i,j} = \alpha_i W_{i,j}^{'} + \beta_i W_{i,j}^{''} + \gamma_i W_{i,j}^{'''}. \quad (2)$$

$W_{i,j}$: the general weight of the semantic cache block j in the node i;
α_i, β_i, γ_i: scalar coefficient, $\alpha_i, \beta_i, \gamma_i \in R$, $\alpha_i, \beta_i, \gamma_i \geq 0$, as $\alpha_i, \beta_i, \gamma_i < 0$ nonsense;
$W_i^{'} = F_i * \frac{C_i}{S_i}$: the semantic weight;
$W_i^{''} = L$: the life (time) weight;
$W_i^{'''} = \frac{1}{2}.|V_k|.((t_x - t_c)^2 + (t_f - t_x)^2)$: the location weight.

The relationship of the semantic, life, and location weight vectors $W_i^{'}$, $W_i^{"}$, and $W_i^{"'}$ are in Nash equilibrium. Each mobile device would want to choose scalar coefficients for its three weight vectors that would result in a cache replacement strategy that would strive to minimize the device's costs in querying for data and maintaining its cache and the time it takes to get queries answered. And, it would be useful to be able to adjust the coefficients separately thus enabling the weighting scheme to be customized for different types of applications and device movement patterns. It is their game and its objective, which is a useful one for caching among mobile devices.

3.2 The Scalar Coefficients of the Weights of the Nodes Satisfied

$$\alpha_i + \beta_i + \gamma_i = 1; 0 < \alpha_i, \beta_i, \gamma_i < 1; \alpha_i, \beta_i, \gamma_i \in R. \tag{3}$$

Theorem 1. *Given three linearly independent vectors* $W_i^{'}$, $W_i^{''}$, *and* $W_i^{'''}$ *and their linear combination* $W_{i,j} = \alpha_i W_{i,j}^{'} + \beta_i W_{i,j}^{''} + \gamma_i W_{i,j}^{'''}$, $\alpha_i, \beta_i, \gamma_i$ *are scalar coefficient,* $\alpha_i, \beta_i, \gamma_i \in R$, $\alpha_i, \beta_i, \gamma_i \geq 0$*: then the following expression is satisfied:*

$$\alpha_i + \beta_i + \gamma_i = 1,\ 0 < \alpha_i, \beta_i, \gamma_i < 1,\ \alpha_i, \beta_i, \gamma_i \in R.$$

Proof: *The rationality of the equation is obvious:*
Let $W_{i,j}^{*} = \alpha_i^{*} W_{i,j}^{'} + \beta_i^{*} W_i^{''} + \gamma_i^{*} W_i^{'''}$, $\alpha_i^{*}, \beta_i^{*}, \gamma_i^{*} \in R$, $\alpha_i^{*}, \beta_i^{*}, \gamma_i^{*} \geq 0$.
$\frac{W_{i,j}^{*}}{\alpha_i^{*}+\beta_i^{*}+\gamma_i^{*}} = \frac{\alpha_i^{*}}{\alpha_i^{*}+\beta_i^{*}+\gamma_i^{*}} W_{i,j}^{'} + \frac{\beta_i^{*}}{\alpha_i^{*}+\beta_i^{*}+\gamma_i^{*}} W_{i,j}^{''} + \frac{\gamma *_i}{\alpha_i^{*}+\beta_i^{*}+\gamma_i^{*}} W_{i,j}^{'''}$.
Then, $W_{i,j} = \frac{W_{i,j}^{*}}{\alpha_i^{*}+\beta_i^{*}+\gamma_i^{*}}$, $\alpha_i = \frac{\alpha_i^{*}}{\alpha_i^{*}+\beta_i^{*}+\gamma_i^{*}}$, $\beta_i = \frac{\beta_i^{*}}{\alpha_i^{*}+\beta_i^{*}+\gamma_i^{*}}$, $\gamma_i = \frac{\gamma_i^{*}}{\alpha_i^{*}+\beta_i^{*}+\gamma_i^{*}}$.
Then $W_{i,j} = \alpha_i W_{i,j}^{'} + \beta_i W_{i,j}^{''} + \gamma_i W_{i,j}^{'''}$, *and* $\alpha_i + \beta_i + \gamma_i = 1$, $0 < \alpha_i, \beta_i, \gamma_i < 1$, $\alpha_i, \beta_i, \gamma_i \in R$.

3.3 The Nash Equilibrium Point of the Semantic, Time and Location Factors

$$\frac{W_i \cdot W_i'}{\|W_i'\|} = \frac{W_i \cdot W_i''}{\|W_i''\|} = \frac{W_i \cdot W_i'''}{\|W_i'''\|}. \tag{4}$$

Theorem 2. *Given three linearly independent vectors,* $W'_i = (w'_{i,0}, \ldots, w'_{i,j}, \ldots, w'_{i,m-1})$, $W''_i = (w''_{i,0}, \ldots, w''_{i,j}, \ldots, w''_{i,m-1})$, $W'''_i = (w'''_{i,0}, \ldots, w'''_{i,j}, \ldots, w'''_{i,m-1})$, *their linear combination* $W_i = (w_{i,0}, \ldots, w_{i,j}, \ldots, w_{i,m-1})$, *and* $W_i = \alpha_i W_i^{'} + \beta_i W_i^{''} + \gamma_i W_i^{'''}$, $\alpha_i, \beta_i, \gamma_i$ *are scalar coefficient,* $\alpha_i, \beta_i, \gamma_i \in r$, $\alpha_i, \beta_i, \gamma_i \geq 0$. *The Nash Equilibrium point of* W'_i, W''_i *and* W'''_i *is*

$$\frac{W_i \cdot W_i'}{\|W_i'\|} = \frac{W_i \cdot W_i''}{\|W_i''\|} = \frac{W_i \cdot W_i'''}{\|W_i'''\|}.$$

Proof: *Because* $W'_i = (w'_{i,0}, \ldots, w'_{i,j}, \ldots, w'_{i,m-1})$, $W''_i = (w''_{i,0}, \ldots, w''_{i,j}, \ldots, w''_{i,m-1})$, $W'''_i = (w'''_{i,0}, \ldots, w'''_{i,j}, \ldots, w'''_{i,m-1})$, *their linear combination* $W_i = (w_{i,0}, \ldots, w_{i,j}, \ldots, w_{i,m-1})$, *and* $W_i = \alpha_i W_i^{'} + \beta_i W_i^{''} + \gamma_i W_i^{'''}$ *(as been shown in Fig. 2).*

Because of $cos\theta_1 = \frac{W_i \cdot W_i'}{\|W_i\| \cdot \|W_i'\|}$, $cos\theta_2 = \frac{W_i \cdot W_i''}{\|W_i\| \cdot \|W_i''\|}$, $cos\theta_3 = \frac{W_i \cdot W_i'''}{\|W_i\| \cdot \|W_i'''\|}$, *we want to find the Nash Equilibrium point to make* $\theta_1 = \theta_2 = \theta_3$, *then* $cos\theta_1 = cos\theta_2 = cos\theta_3$.

In other words, the Nash Equilibrium point among W_i', W_i'' *and* W_i''' *should be the point that has an equal correlation coefficient to each of the vectors. In other words* $\frac{W_i \cdot W_i'}{\|W_i\| \cdot \|W_i'\|} = \frac{W_i \cdot W_i''}{\|W_i\| \cdot \|W_i''\|} = \frac{W_i \cdot W_i'''}{\|W_i\| \cdot \|W_i'''\|}$, *then we get:* $\frac{W_i \cdot W_i'}{\|W_i'\|} = \frac{W_i \cdot W_i''}{\|W_i''\|} = \frac{W_i \cdot W_i'''}{\|W_i'''\|}$.

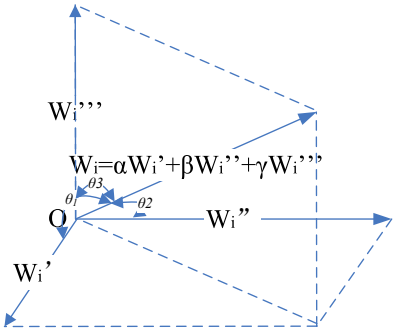


Fig. 2. The relationship of W_i', W_i'', W_i''' and W_i

Combining (2) and (3), the paper builds the following linear binary simple equations:

$$\begin{cases} \alpha_i + \beta_i + \gamma_i = 1 \\ \frac{W_i \cdot W_i'}{\|W_i'\|} = \frac{W_i \cdot W_i''}{\|W_i''\|} = \frac{W_i \cdot W_i'''}{\|W_i'''\|} \end{cases}.$$

For $W_i = \alpha_i W_i' + \beta_i W_i'' + \gamma_i W_i'''$,
$\frac{(\alpha_i W_i' + \beta_i W_i'' + \gamma_i W_i''') \cdot W_i'}{\|W_i'\|} = \frac{(\alpha_i W_i' + \beta_i W_i'' + \gamma_i W_i''') \cdot W_i''}{\|W_i''\|} = \frac{(\alpha_i W_i' + \beta_i W_i'' + \gamma_i W_i''') \cdot W_i'''}{\|W_i'''\|}$,
then $(\alpha_i W_i' + \beta_i W_i'' + \gamma_i W_i''') \cdot W_i' \cdot \|W_i''\| \cdot \|W_i'''\|$
$= (\alpha_i W_i' + \beta_i W_i'' + \gamma_i W_i''') \cdot W_i'' \cdot \|W_i'\| \cdot \|W_i'''\|$
$= (\alpha_i W_i' + \beta_i W_i'' + \gamma_i W_i''') \cdot W_i''' \cdot \|W_i'\| \cdot \|W_i''\|$.
For $\alpha_i + \beta_i + \gamma_i = 1$, then

$$\alpha_i = \frac{\|W_i''\| + \|W_i'''\|}{2(\|W_i'\| + \|W_i''\| + \|W_i'''\|)} = \frac{\sqrt{\sum_{j=0}^{m-1} w_{i,j}''^{\,2}} + \sqrt{\sum_{j=0}^{m-1} w_{i,j}'''}}{2\left(\sqrt{\sum_{j=0}^{m-1} w_{i,j}'^{\,2}} + \sqrt{\sum_{j=0}^{m-1} w_{i,j}''^{\,2}} + \sqrt{\sum_{j=0}^{m-1} w_{i,j}'''^{\,2}}\right)};$$

$$\beta_i = \frac{\|W_i'\| + \|W_i'''\|}{2(\|W_i'\| + \|W_i''\| + \|W_i'''\|)} = \frac{\sqrt{\sum_{j=0}^{m-1} w_{i,j}'^{\,2}} + \sqrt{\sum_{j=0}^{m-1} w_{i,j}'''}}{2\left(\sqrt{\sum_{j=0}^{m-1} w_{i,j}'^{\,2}} + \sqrt{\sum_{j=0}^{m-1} w_{i,j}''^{\,2}} + \sqrt{\sum_{j=0}^{m-1} w_{i,j}'''^{\,2}}\right)};$$

$$\gamma_i = \frac{\|W_i'\| + \|W_i''\|}{2(\|W_i'\| + \|W_i''\| + \|W_i'''\|)} = \frac{\sqrt{\sum_{j=0}^{m-1} w_{i,j}'^{\,2}} + \sqrt{\sum_{j=0}^{m-1} w_{i,j}''}}{2\left(\sqrt{\sum_{j=0}^{m-1} w_{i,j}'^{\,2}} + \sqrt{\sum_{j=0}^{m-1} w_{i,j}''^{\,2}} + \sqrt{\sum_{j=0}^{m-1} w_{i,j}'''^{\,2}}\right)};$$

so: $0 < \alpha_i, \beta_i, \gamma_i < 1, \alpha_i, \beta_i, \gamma_i \in R$.

4 Algorithms for Nash Equilibrium Semantic Cache Schemes

This section discusses the concrete implementation of the algorithms. Our algorithm is consisted of three sub-algorithms:

1) Network Manager Algorithm: The Network Manager Algorithm manages the network among the clients and the servers, which includes three functions: Node Move Function, Building Routing Table Function, and Searching Routing Table Function.

2) Query Processor Algorithm: Query Processor Algorithm generates queries and processes queries via the semantic caches and servers. It randomly generates a query that is a data structure. Afterward, firstly it queries to locate the node, then it queries in the neighboring node and finally it queries to the server.

3) Semantic Cache Manager Algorithm: The core part of our work focus on the Semantic Cache Manager Algorithm, which calculates semantic cache scalar coefficients, manages the store and replaces of the semantic cache. First, it calculates the Semantic Weight, Time Weight, and Location Weight of every semantic-cache block using the basic information about frequency, size, cost, time, and location. Second, it finds the inserting place and calculates the store space. Finally if the space is large enough, then store, otherwise replace.

5 Performance Evaluation

We examine the performance of the proposed scheme in a mobile computing environment through a simulation study. Our simulator is developed by ourselves and implemented in C++, which simulates a simple but typical mobile computing model. Then, we describe the conducted experiments, and further analyze the results. The simulation experiments are executed on an IBMx255 server that runs Linux with four Intel Xeon MP 3.0GHz/400MHz processors and 18G DDR main memory.

5.1 Simulation Environment

In the simulation environment, we set two servers and multiple mobile clients, which use a wireless link among them. They move in the range $(X1, X2, Y1, Y2)$. We assume that at least one or two servers maintain a complete copy of the database and act as a sink and a database server. The queries are submitted by the different mobile nodes in different situations, including the location, time and requirement. The semantic cache is located on the mobile nodes. First a query is processed locally; second the query is sent to the neighboring nodes for processing, when it cannot be answered via the cache; finally, it is sent to the server nodes, when it cannot be answered via the neighboring nodes.

Mobile Clients Model: The mobile clients are simulated to be composed of the following modules: network manager, which manages the network among the mobile nodes and the central servers; query processor, which generates and

processes queries via the semantic cache; and semantic cache manager, which calculates the different parameters, and manages the store and replacement of the semantic cache.

System Parameters: Table 1 gives the parameters that specify the physical resources of the modeled mobile system. All of the devices move within the spatial domain according to the random method, a Grid Configuration from 50×100 to 1000×2000. We vary the Number of Mobile Devices from 100 to 5000. The Wireless Network Bandwidth is given by Bandwidth, which has a typical value of $19.2kbps$. The Wireless Radius is changing from 10 to 100, most of time focusing on 100. We set the Maximum Hops to forward a routing message to 2. This value was chosen by experiment because it attained good cache effects, but did not incur high additional costs. Database relations, cached semantic segments and cache maintenance data are all physically stored in pages, whose size is specified by Server Pages, which is set to 1000000. Client Cache defines the size of the memory cache at the client side, which is set 10000.

Initially, all of the data are stored in the simulated servers, and no device stores any data in local storage. During the Simulation Period T, which is changing from $100s$ to $5000s$, the system randomly generates one mobile device to issue 1 query, which is a different type, price, and time, according to the location of the mobile device, in every time t. We vary the Number of Mobile Devices from 100 to 5000, which yields a moderate-scale data set [7]. The Speed Range of the mobile devices is from $1unit/s$ to $10unit/s$, including of PERSON and CAR. Because usually the simulation period T is very long, we choose Step Time, which is changes from $50s$ to $200s$ to calculate the relative parameters, for example: Average Response Time and Average Local/Peer Hit Ratio.

Table 1. Parameters Used in Simulation

Parameter	Setting
Grid Configuration	$50 \times 100, ..., 1000 \times 2000$
Number of Mobile Devices	$100, ..., 1000$
Wireless Network Bandwidth	$19.2kbps$
Wireless Radius	$10, ..., 100$
Max Hops	2
Speed Range	$1unit/s - 10unit/s$
Simulation Period	$100s, ..., 5000s$
Step Time	$50s, 100s, ..., 200s$
Client Cache Size	10000
Server Page Size	1000000
Semantic_degree	$1, 2, 3, 4$
Location_degree	$1, ..., 10$

Here, we introduce one parameters: Semantic_degree: $(1, 2, 3, 4)$ means the query generates a rate of semantics, for example, there are three query types: Restaurant, Hotel and Parking_lots. We consider random to skewed distributions:

1, 33% of Restaurant: 33% of Hotel: 33% of Packing_lots;
2, 50% of Restaurant: 30% of Hotel: 20% of Packing_lots;
3, 70% of Restaurant: 20% of Hotel: 10% of Packing_lots;
4, 80% of Restaurant: 10% of Hotel: 10% of Packing_lots.

We consider two main performance aspects:

(1) Average Response Time. The response time is defined as the elapsed simulation time from the moment that a query is issued at a mobile device M_{org} to the moment that M_{org} obtains all of the answers.

(2) Average Peer Hit Ratio. A storage hit occurs when a desired semantic cache block is found, if the system hasn't retrieved it from the server. We use the peer hit ratio for the percentage found in peer storage.

5.2 Experiments

Because the strategy of using CSC is obviously poorer than the other approaches, we mostly compared the following three different strategies: $SCSTL$, $B\&B_{GDSF}$, $CSDS$.

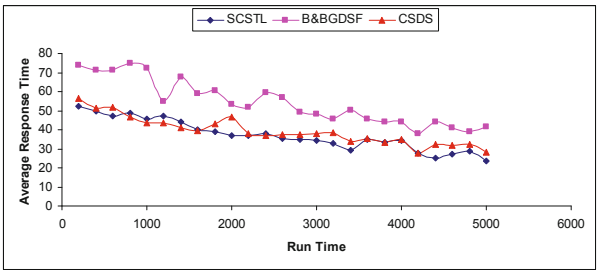
We chose two parameters to compare the simulation result, which are Semantic_degree: The Semantic_degree takes on the values of $1, 2, 3, and 4$.

(1) Average Response Time. In Fig. 3, when the Semantic_degree is 1 the approach $CSDS$ becomes similar to our approach $SCSTL$. For all other situations, the Average Response Time of our approach $SCSTL$ was always obviously better than the others.

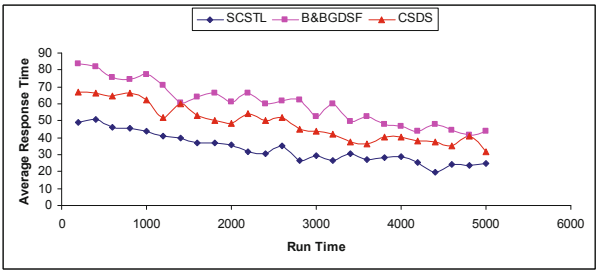
(2) Average Peer Hit Ratio. In Fig. 4, in the beginning, when the Semantic_degree is 1, the approach $SCSTL$ is poorer than the others. However, when the Semantic_degree increase to 2 and 3, the efficiency increases. When the Semantic_degree reaches a max value of 4, our approach $SCSTL$ attains the best efficiency of the Average Peer Hit Ratio. In daily living, people will choose some hot places, which means that our approach can attain the best efficiency in a realistic application. Moreover, generally speaking, in the beginning, the Average Peer Hit Ratio of our approach $SCSTL$ is not as good as the others, but as time goes on, its efficiency increases, and in the end it obtains the best result, which means that our approach $SCSTL$ is specifically suitable for a long-term application.

Furthermore, higher Average Peer Hit Ratio, means less communication and computation cost. Especially in realistic applications, communication distance between peer to peer is much less than between peer to server, maybe 100 times or more, so it can save lots of bandwidth and energy. Moreover, it is obviously that computation and query costs in peer are much less than in server, maybe 1000 times or more, for the cache data quantity is limited. So that, in realistic applications, $SCSTL$ can attain much better efficiency than others.

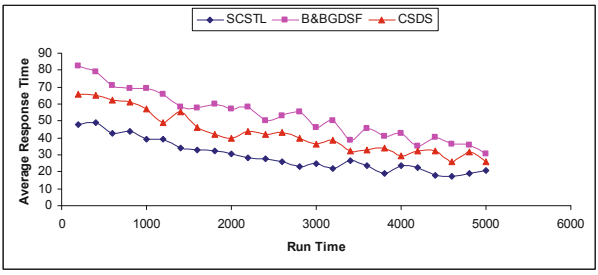
The reason for all of the better efficiencies of the Average Response Time and Average Peer Hit Ratio is that our approach $SCSTL$ synthetically considers all of the semantic, time and location factors.



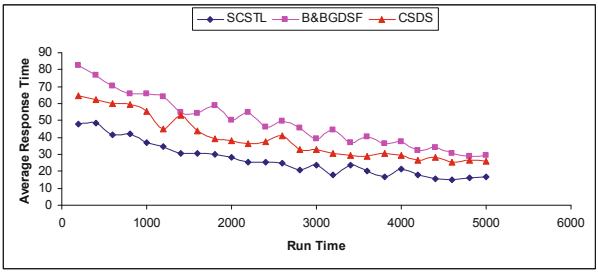
(a)



(b)

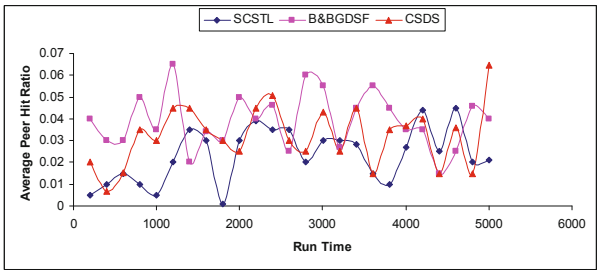


(c)

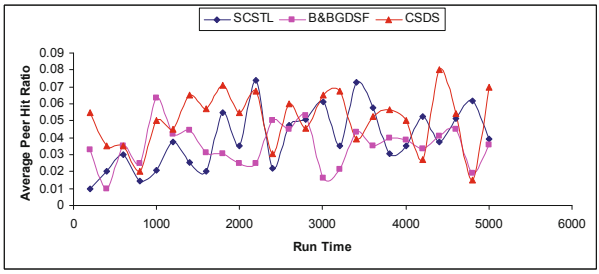


(d)

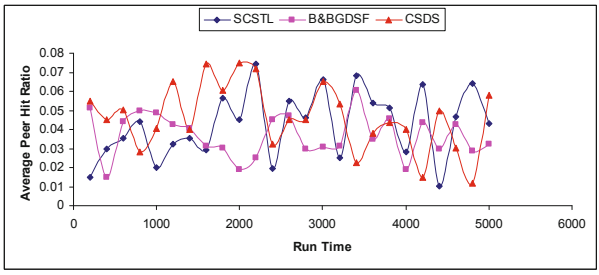
Fig. 3. The simulation results for the Average Response Time of the following approaches: $SCSTL$, $B\&B_{GDSF}$, and $CSDS$: (a), (b), (c), (d) means the Location_degree focus on $CSDS$ and the Semantic_degree takes on the values of $1, 2, 3, 4$



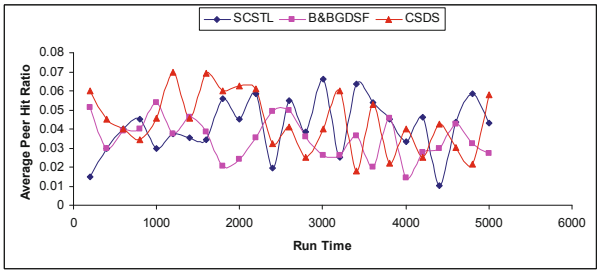
(a)



(b)



(c)



(d)

Fig. 4. The simulation results for the Average Peer Hit Ratio of approach $SCSTL$, $B\&B_{GDSF}$, $CSDS$: (a), (b), (c), (d) means the Location_degree focus is on 3, while the Semantic_degree takes on the values of $1, 2, 3, 4$

6 Conclusions and Future Work

Conclusions: To improve the query efficiency of a sensor grid, we propose our semantic cache scheme, which includes the following: (i) Based on the Nash Equilibrium scheme, an application of three scalar coefficients is proposed in term of the analysis of the relationship among semantic, time and location factors. (ii) After that, we specialize and summarize the general Nash Equilibrium scheme utilizing the equal correlation coefficient among the new and old vectors to find the point of Nash equilibrium and resolves the scalar coefficients. (iii) It uses these scalar coefficient to attain an optimum cost model of the semantic cache, for query optimization in the context of distributed query processing. (iv) We emphasize that this method can extend to any finite-dimension, which are other schemes cannot do.

Future Work: To extend to the N-vectors correlation. All of author's research is in organic architecture. After discussing three-dimensional vectors correlations, the algorithm can be extended to infinite N-vectors correlations.

References

1. Ewen, S., Schelter, S., Tzoumas, K., Warneke, D., Markl, V.: Iterative parallel data processing with stratosphere: an inside look. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 1053–1056 (2013)
2. Huang, Z., Jensen, C.S., Ooi, B.C.: Collaborative spatial data sharing among mobile lightweight devices. In: Papadias, D., Zhang, D., Kollios, G. (eds.) SSTD 2007. LNCS, vol. 4605, pp. 366–384. Springer, Heidelberg (2007)
3. Li, Y., Jin, D., Hui, P., Su, L., Zeng, L.: Revealing contact interval patterns in large scale urban vehicular ad hoc networks. In: Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (2012)
4. Ren, Q., Dunham, M.H.: Using clustering for effective management of a semantic cache in mobile computing. In: Mobile 1999 Proceedings of the 1st ACM International Workshop on Data Engineering for Wireless and Mobile Access (1999)
5. Savary, L., Gardarin, G., Zeitouni, K.: Geocach a cache for gml geographical data. International Journal of Data Warehousing & Mining 3, 66–87 (2007)
6. Villas, L.A., Boukerche, A., Araujo, R., Loureiro, A.A.: A spatial correlation aware algorithm to perform efficient data collection in wireless sensor networks. Ad Hoc Networks 12, 69–85 (2014)
7. Xu, B., Vafaee, F., Wolfson, O.: In-network query processing in mobile p2p databases. In: Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (2009)

Exploiting Diversification in Gossip-Based Recommendation*

Maximilien Servajean[1], Esther Pacitti[1], Miguel Liroz-Gistau[2], Sihem Amer-Yahia[3], and Amr El Abbadi[4]

[1] INRIA & LIRMM, University of Montpellier, France
{servajean,pacitti}@lirmm.fr
[2] INRIA & LIRMM, Montpellier, France
miguel.liroz_gistau@inria.fr
[3] CNRS, LIG
sihem.amer-yahia@imag.fr
[4] Dpt. of Computer Science, University of California at Santa Barbara
amr@cs.ucsb.edu

Abstract. In the context of Web 2.0, the users become massive producers of diverse data that can be stored in a large variety of systems. The fact that the users' data spaces are distributed in many different systems makes data sharing difficult. In this context of large scale distribution of users and data, a general solution to data sharing is offered by distributed search and recommendation. In particular, gossip-based approaches provide scalability, dynamicity, autonomy and decentralized control. Generally, in gossip-based search and recommendation, each user constructs a cluster of "relevant" users that will be employed in the processing of queries. However, considering only relevance introduces a significant amount of redundancy among users. As a result, when a query is submitted, as the user profiles in each user's cluster are quite similar, the probability of retrieving the same set of relevant items increases, and recall results are limited. In this paper, we propose a gossip-based search and recommendation approach that is based on a new clustering score, called *usefulness*, that combines relevance and diversity, and we present the corresponding new gossip-based clustering algorithm. We validate our proposal with an experimental evaluation using three datasets based on *MovieLens*, *Flickr* and *LastFM*. Compared with state of the art solutions, we obtain major gains with a three order of magnitude recall improvement when using the notion of *usefulness* regardless of the relevance score between two users used.

1 Introduction

In the context of Web 2.0, users become massive producers of diverse data (*e.g.* photos, videos, scientific data) that can be stored in a large variety of systems (*e.g.* DropBox, Facebook, Flickr, Google+, local computer or smartphone). Users

* Work conducted within the Institut de Biologie Computationnelle and partially funded by the labex NUMEV and the CNRS project Mastodons.

A. Hameurlain et al. (Eds.): Globe 2014, LNCS 8648, pp. 25–36, 2014.

are often willing to share their data with other users in a community of interest. However, the fact that their data spaces are distributed in many different systems makes data sharing especially difficult. For instance, an artist photographer who wants to share her pictures within an online community of photographers may have to log in several different Web applications such as deviantArt, Facebook or Flickr, each with a different interface and account. Similarly, a scientist who needs to search for scientific datasets within an online community of scientists will be faced with the problem that the relevant data is typically distributed in many different labs' servers or scientists' local computers. Furthermore, since this data is hidden to web crawlers, traditional search engines become useless. In order to mitigate this problem, some Web applications allow grouping several accounts and data from different systems (*e.g.* Facebook enables to regroup DropBox and blogs into a single Facebook account). However, they are limited to a few well-known systems.

In this context of large scale distribution of users and data, a general solution to data sharing is offered by distributed search and recommendation [1, 2]. In this paper, we adopt a peer-to-peer gossip-based approach, because it provides important properties such as scalability, dynamicity, autonomy and decentralized control. Within an online community, each user u is associated to a virtual data space that contains all the data items (stored in different systems) it shares. Given u and a keyword query q, the goal of our search and recommendation approach is to recommend to u items that are relevant with respect to q and that are shared by other users, regardless of the systems that store the items. Then, a recommended item is simply a reference that can be used to retrieve the actual data item. In other words, we combine search and recommendation in the sense that a user u searches relevant items among those recommended by users similar to u.

Distributed search and recommendation has received considerable attention [1–4]. However, one open problem is the ability to attain high recall results. A query is generally forwarded only to a subset of users who will be employed to process queries and return recommendations. To compute this subset of users, many solutions cluster relevant user profiles implicitly using gossip protocols. Gossip protocols are known to be highly resilient, scalable and converge quickly [5], which makes them a good alternative for distributed search and recommendation. A *User Network* (*U-Net* in the following) refers to the cluster of relevant users, a user u is aware of by gossiping, using a score (*e.g.* similarity between u and the users in *U-Net*). At each gossip round, the most relevant users are kept in *U-Net*. Since *U-Net* is used to guide recommendations given a keyword query, the relevance score used in the clustering process plays a very important role to increase the number of relevant items retrieved with respect to the whole set of items (*i.e.* recall), known as the global corpus.

Relevance scores (*e.g.* Jaccard, overlap) define how well a user profile v meets the needs of another user u. Most of the existing solutions exploit different kinds of relevance scores to increase recall [2–4, 6, 7]. But recall results remain limited.

The reason why recall remains limited is because using relevance as the clustering score introduces a significant amount of redundant user profiles in *U-Net*. As a result, when a query is submitted, since many user profiles in *U-Net* are quite similar (*i.e.* redundant), and these users are chosen to provide recommendations to answer the query, the probability of retrieving the same set of relevant items increases and recall results remain low. In *Information Retrieval*, usefulness is used as a way to overcome redundancy between the items of a result list by combining relevance with diversity [8, 9]. In our context, we claim that usefulness can be used when clustering user profiles in *U-Net*, instead of just relevance. This way, a more diverse set of results will be returned from different users and the recall would be enhanced.

In this paper, we propose a gossip-based search and recommendation approach based on a new clustering score, called usefulness, that combines relevance and diversity. As we show experimentally, this new score is able to increase significantly the quality of the recommendations returned by the system. However, existing peer-to-peer clustering algorithms are no longer suitable since they are optimized for relevance only. Therefore we also propose a new clustering algorithm especially conceived for usefulness.

In summary, we make the following contributions:

1. We show that *usefulness* is a good way to increase recall and that it should be expressed as a known probabilistic diversification score [8, 9].
2. We propose a clustering algorithm that maintains a useful *U-Net* over a gossip overlay using the usefulness score.
3. We validate our approach with an experimental evaluation using three different datasets: *MovieLens*, *Flickr* and *LastFM*. We observe that diversification enables a huge increase of recall regardless of the relevance score used. Compared with state of the art solutions, we obtain an excellent gain with recall results up to three times better when using the notion of usefulness.

This paper is organized as follows. Section 2 provides some basic concepts and gives the problem definition. In Section 3, we describe our new clustering score and present in details the new clustering algorithm that maintains *U-Net*. In Section 4, we provide an experimental evaluation. In Section 5, we compare our contributions with related work. Finally, Section 6 concludes and provides directions for future work.

2 Basic Concepts and Problem Definition

In this section, we introduce the background necessary to understand the problem we address. In our distributed search and recommendation approach, whenever a user u submits a query q, the system sends q to a subset of users that we call *U-Net*, and who will return their relevant results to u and will also recursively forward the query to the users in their *U-Net* until the *TTL* is reached. To build *U-Net*, we use a two steps approach. First, based on *random gossiping* each user u is aware of other peers available on the network. Second, by means of a *clustering* algorithm, u chooses among these users the best ones to answer u's queries and keep them in *U-Net*.

More precisely, our peer-to-peer model is expressed based on a graph $G = (U, I, E)$, where $U = \{u_1, ..., u_n\}$ is the set of users distributed over the network, $I = \{i_1, ..., i_m\}$ the set of shared data items (in the following, an item refers to a data item), and $E = \{e_1, ..., e_k\}$ the set of directed edges among users and between users and items. This model is very generic. In our case, users are independent nodes in the network. A node can be a physical computer or a virtual node in a server.

Definition 1 (*U-Net*). *Given a user u, its User Network, or U-Net, refers to the cluster of relevant users u is aware of. There is an edge $e(u, v)$ in the graph between u and a user v, if v is in u's U-Net.*

With random gossiping [5], each user keeps locally a random view of its dynamic acquaintances (or view entries). Each view entry corresponds to a user profile. Periodically, each user chooses randomly a contact (view entry) to gossip with. The two involved users then exchange a subset of each others view (*i.e.* user profiles), and update their view state. Then, after each gossip exchange, the random view is used to update the *U-Net* if more relevant profiles are found in the updated view. We use *Jaccard* as the relevance score to select the best users:

$$\mathbf{Jaccard}(\mathbf{u}, \mathbf{v}) = |I_u \cap I_v| \ / \ |I_u \cup I_v| \tag{1}$$

Where I_u and I_v are the items shared by user u and v respectively.

Here, we use the vector space model to represent items and user profiles [10]. Specifically, each item *it* is modelled as a sparse vector containing only the weights of the keywords $k_1, ..., k_z$ in it. The weight of each keyword is computed using $tf \times idf$. Distributed $tf \times idf$ can be easily implemented using gossip protocols. Indeed, the first part of the score, denoted *tf*, can be computed locally, and the second part, denoted *idf*, only needs average information (*e.g.* average number of items per user) to be computed. These averages can be easily computed using gossip protocols [11]. Due to lack of space, we do not develop this protocol in this paper.

Each *user profile* is defined based on the items the user shares, I_u (*i.e.* content based recommendation). We choose a relevance score (*i.e. Jaccard*) that works well with content-based recommendation, but other relevance measures and profiles definition methods could be used as well.

As mentioned before, whenever a user u submits a keywords query $q = k_1, ..., k_w$, the query is redirected to all users in the participating users' *U-Net* recursively, until a predefine upper threshold, TTL (*i.e. Time-To-Live*). Whenever a user v receives a query, it computes its *top-k* most relevant items with respect to the query using a specific relevance score (*e.g.* Jaccard). Then, v returns them to u. A recommended item is defined by its identifier, its $tf \times idf$ vector, v's identifier and v's profile. Once u receives the set of recommended items from $v_1, ..., v_n$ with respect to its query q, it ranks them based on their relevance with respect to the query:

$$Rec_q = rank(rec_q^1(it_1, ...) \cup ... \cup rec_q^n(it_p, ...)) \tag{2}$$

Where $rec_q^1(it_1, ...)$ is a recommendation (*i.e.* a set of recommended items) coming from a user v_1.

To evaluate the quality of search and recommendation, we use the *recall* measure [12]. Recall captures the fraction of items that have been successfully recommended: $recall = |Iret_q| \ / \ |Irel_q|$, where $Iret_q \in I$ refers to the relevant items recommended with respect to a query q, and $Irel_q \in I$ refers to all the relevant items with respect to query q.

Problem Definition: Given a user $u \in U$, a query q, I in G, and a gossip based overlay, the goal is to maximize the number of relevant items with respect to q returned to u while minimizing TTL.

3 Diversified Clustering and Algorithm

In this section, we show that usefulness is an excellent way to increase recall results of gossip-based recommendation, and can be used as a clustering score. In section 3.1, we formally show that to increase recall, usefulness should take into account relevance and diversity. Next, Section 3.2 presents the *Useful U-Net* clustering algorithm deployed over a gossip-based overlay.

3.1 Usefulness Score

The usefulness score should be designed such that it maximizes the probability that a user u can retrieve relevant items given a random query q, known as the coverage probability. In other words, u's neighbors $v_1, ..., v_n \in G$ should be chosen such that the number of relevant items (with respect to the queries u will submit) that can be accessed through them is maximized.

Let Q be the set of all possible queries (all the combinations of terms), and $P(Q_v)$ the probability that a user v can return at least one relevant item given a random query $q \in Q$. In the following, we first define the coverage with respect to $U\text{-}Net_u = \{v_1, ..., v_n\}$. Then, based on coverage, we express the usefulness of a user v with respect to the other users in u's *U-Net*.

Definition 2 (Coverage). *Given Q and $U\text{-}Net_u = \{v_1, ..., v_n\}$, the users in u's U-Net. The coverage is the probability that at least one of the user in u's U-Net can return at least one item given a random query $q \in Q$. Coverage is denoted $P(Q_{v_1} \cup Q_{v_2} \cup ... \cup Q_{v_n})$.*

The user profiles $v_1, ..., v_n$ must be selected such that the coverage probability is maximized. Formula 3 develops the coverage probability with respect to every user in u's *U-Net*.

$$P(Q_{v_1} \cup ... \cup Q_{v_n}) = \sum_{j \in 1,...,n} (P(Q_{v_j}) - P(Q_{v_j} \cap (Q_{v_{j+1}} \cup ... \cup Q_{v_n}))) \qquad (3)$$

$P(Q_{v_j}) - P(Q_{v_j} \cap (Q_{v_{j+1}} \cup ... \cup Q_{v_n}))$ represents the coverage added by user v_j with respect to the users $v_{j+1}, ..., v_n$. As a consequence, when $j = n$, only $P(Q_{v_j})$ is considered as there is no more user profiles to compare with.

In the following, we define the usefulness of a user profile v_i with respect to the coverage probability.

Definition 3 (Usefulness). *Given u's U-Net, the usefulness of a user profile v_j is the probability that it can return relevant items for a random query q, that*

could not be returned by other users in u's U-Net. In other words, it is defined as follows:

$$usefulness(v_j|v_{j+1},...,v_n) = P(Q_{v_j}) - P(Q_{v_j} \cap (Q_{v_{j+1}} \cup ... \cup Q_{v_n})) \tag{4}$$

Formula 4 shows that the usefulness score should consider relevance $P(Q_{v_j})$ and take into account $P(Q_{v_j} \cap (Q_{v_{j+1}} \cup ... \cup Q_{v_n}))$ which corresponds to the redundancy of user profile v_j with respect to the other user profiles $v_{j+1}, ..., v_n$.

In the following, we show that $usefulness(v_j|v_{j+1},...,v_n)$ can be expressed into a known probabilistic diversification model [8, 9]. In Formula 5 we first integrate usefulness (the right hand side of Formula 4) into a conditional probability.

$$\begin{aligned} P(Q_{v_j}) - P(Q_{v_j} \cap (Q_{v_{j+1}} \cup ... \cup Q_{v_n})) &= P(Q_{v_j}) \times (1 - P(Q_{v_{j+1}} \cup ... \cup Q_{v_n}|Q_{v_j})) \\ &= P(Q_{v_j}) \times P(\bar{Q}_{v_{j+1}} \cap ... \cap \bar{Q}_{v_n}|Q_{v_j}) \end{aligned} \tag{5}$$

Similar to [8, 9, 13], we assume that the redundancy of a user profile v_1 with another user profile v_2 is independent of its redundancy with other users and we derive Formula 6.

$$P(Q_{v_j}) \times P(\bar{Q}_{v_{j+1}} \cap ... \cap \bar{Q}_{v_n}|Q_{v_j}) = P(Q_{v_j}) \times \prod_{i \in j+1,...,n} (1 - P(Q_{v_i}|Q_{v_j})) \tag{6}$$

Finally, we observe that the usefulness of a user profile is clearly similar to the probabilistic diversification problem used in [8, 9] and presented in Formula 7.

$$usefulness(v_j|v_{j+1},...,v_n) = rel(v_j) \times \prod_{i \in j+1,...,n} (1 - red(v_j, v_i)) \tag{7}$$

where $rel(v_j) = P(Q_{v_j})$ is the relevance of user profile v_j and $red(v_j, v_i) = P(Q_{v_i}|Q_{v_j})$ is the redundancy of user profile v_j with respect to the other user profile v_i.

3.2 Useful U-Net Clustering

We now present in details our clustering algorithm that maintains a useful *U-Net* over a random gossip overlay using the usefulness score.

Given the set of users in the random view, the goal of the clustering algorithm is to compute the usefulness of each user found in the view, with respect to those that were previously added to the *U-Net*, taking into account relevance and diversity, as defined in Equation 7, and to update the *U-Net* as consequence.

Based on random gossiping [5], each user u maintains a set of random view entries corresponding to the users profile u is aware of. Periodically, users gossip, and exchange a random subset of their views entries. After the random gossip merging phase, the clustering algorithm, which corresponds to the *Useful U-Net* Algorithm depicted in Algorithm 1, is triggered. In fact, taking into account the previous gossip exchange, the algorithm selects the most useful users from the random view considering the useful users previously selected (*i.e.* from the previous gossip rounds) in the *U-Net*. The algorithm uses three main data structures: random view, *U-Net*, and the candidate list. The random view and the *U-Net* are initialized when u joins the network, and continuously updated as a result of random gossip. The candidates list contains the user profiles that will potentially be

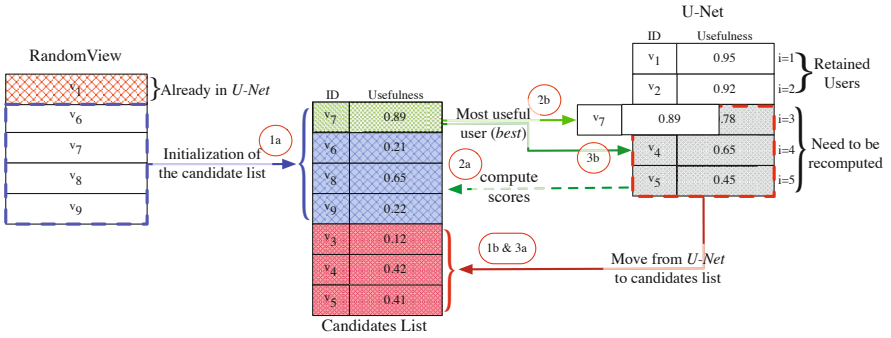


Fig. 1. An example of the execution of Useful-Unet

added to the *U-Net* and is initialized each time the clustering algorithm is triggered.

In the following we present in more details the *Useful U-Net* algorithm based on the example of Figure 1. The random view entries correspond to the profiles of users v_1, v_6, v_7, v_8, v_9. The previous useful user profiles are v_1, v_2, v_3, v_4, v_5 and are stored in *U-Net*. Assuming that the algorithm is executed in u's node, the algorithm input is u's profile, its random view denoted $RandomView_u$ and its *U-Net* denoted $U\text{-}Net_u$. The data structure used for *U-Net* is an array of size N of user profiles, associated to their usefulness score and sorted in decreasing order of usefulness. The output of the algorithm is the updated *U-Net*. *Useful U-Net* algorithm has three main parts:

1. The first part (lines 1 to 6) finds the *best* useful user profile from the random view, and the position i where it should be inserted in the *U-Net* (recall that the usefulness score of a user depends on its position in the *U-Net*). As a consequence, the update of the *U-Net* will only concern the user profiles from position i to N. To find the *best* useful user from the random view, the algorithm first initializes the *candidates* list with all users in the random view except those already in the *U-Net* (line 2). In Figure 1, v_1 is already in the *U-Net*, so the candidates list is initialized with the users v_6, v_7, v_8, v_9 ($1a$). For each position i in *U-Net*, all the usefulness scores of the candidates are computed using Formula 7 taking into account the set of users in the *U-Net* at positions $1, ..., i-1$, and compared with the usefulness score of the user profile in $U\text{-}Net_u[i]$. If the best user profile in *candidates* is more useful than $U\text{-}Net_u[i]$, then, the algorithm stops iterating (line 6). If there is more than one best user profile, the best user profile is chosen randomly with respect to the set of best user profiles. In Figure 1, v_7 is more useful than v_3 at the third position in u's *U-Net* because v_3's usefulness is 0.78 while v_7's usefulness is 0.89 ($1b$). If there is no user profile in the candidate list whose profile score is superior to any user profile in the *U-Net*, position N is reached and the algorithm stops. Only the scores of the user profiles up to position i are definitive. Thus, in our example, the scores of v_4, v_5, v_6, v_8, v_9 are not definitive because they are either not in the *U-Net* or after i.

2. The second part (lines 7 to 10) copies and deletes the remaining user profiles (from position i to N) from the *U-Net* to the candidates ($2a$) list because their scores need to be recomputed using Formula 7 and with respect to the best user profile in *candidates* (computed in part 1). Then, the best user profile is inserted in position i. In the on-going example of Figure 1, the user profiles v_3, v_4, v_5 are copied and removed from the *U-Net* to the candidates list and user profile v_7 is added in the *U-Net* at position 3 ($2a$ and $2b$).
3. Finally, in the last part (lines 11 to 15), the algorithm iteratively computes, for each empty position i in the *U-Net* (positions emptied in part 2), the scores of the user profiles in the candidates list using Formula 7 and taking into account the set of users in the *U-Net* at positions $1, ..., i-1$ (lines 12 and 13 and step $3a$ in the figure). Then, the most useful candidate is moved to the *U-Net* at that position (line 15 and step $3b$ in the figure). The algorithm repeats these steps until all the positions in *U-Net* are filled out (line 11).

Recall that gossip protocols converge quickly [3]. As a consequence the *U-Net* will also converge quickly and, in general, tends to stabilize. Therefore, the algorithm will stop at step $1b$ more and more frequently.

Algorithm 1. Useful *U-Net*

Input: u profile, $U\text{-}Net_u$ (array[1..N]), $RandomView_u$
Output: $U\text{-}Net_u$ is updated with respect to the $RandomView$

```
1  candidates : unsorted list of user profiles;
2  candidates ← RandomView_u − U-Net_u; best ← ∅; i ← 0;
3  repeat
4  |  i++;
5  |  for each c_j ∈ candidates do score(c_j) ← usefulness(c_j,u,U-Net_u[1..i − 1])
   |  best ← arg max_{c∈candidates}(score(c));
6  until i=N or score(best) > score(U-Net[i]);
7  if score(best) > score(U-Net[i]) then
8  |  after← U-Net_u[i..N]; U-Net_u[i] ← best; i++;
9  |  candidates ← candidates − best;
10 |  candidates ← after ∪ candidates; U-Net_u ← U-Net_u − after;
11 |  while i < N and candidates≠ ∅ do
12 |  |  for each c_j ∈ candidates do
13 |  |  |  score(c_j) ← usefulness(c_j, u, U-Net_u[1..i − 1]);
14 |  |  best ← arg max_{c∈candidates}(score(c)); U-Net_u[i] ← best;
15 |  |  candidates ← candidates − best; i++;
```

4 Experimental Evaluation

In this section, we provide an experimental evaluation to validate our approach and compare it to other state-of-the-art solutions. We conducted a set of experiments using three datasets which correspond to *MovieLens*, *Flickr* and *LastFM*. In Section 4.1, we introduce the experimental setup of our evaluation. Then, in Section 4.2, we present and discuss the experimental results.

4.1 Experimental Setup

We ran our experiments on the *PeerSim* simulator[1]. We used three different datasets: *MovieLens*, *Flickr* and *LastFM*. *MovieLens* dataset is composed of users that rated movies. *Flickr* dataset is composed of users that submitted or added a picture to their favorites. Each user also associates tags to the pictures he/she submits. Finally, LastFM dataset is composed of users who listen and associate tags to artists. Each dataset has different features, in particular users are more or less redundant if the number of items per user is more or less respectively. The characteristics of the datasets are summarized in the following table.

dataset	items	# items	# users	avg items/user
MovieLens	Movies	3,900	6,040	166
Flickr	Pictures	2,029	2,000	3.7
LastFM	Artists	23,346	2,000	98

The queries used in the experiments consist of: In *MovieLens*, for each user, a random subset of movies are shared and the rest are used as the queries to submit. In particular, the words in the title are used as separate keywords. In *Flickr* and *LastFM* queries are computed as the random association of several tags submitted by a given user on a given item. An experiment is composed of two parts. First, all users gossip during 400 rounds until convergence. Then, every 20 gossip rounds all users submit one of their queries. The experiment stops at 500 gossip rounds. We measure the average recall results. The recall enables to compute the fraction of items that has been successfully recommended as presented in Section 2. On the *MovieLens* dataset, the recall value is 1 if the movie has been found and 0 otherwise. On *Flickr* and *LastFM*, the recall is the proportion of pictures in the whole dataset that contains all query's keywords that have been returned to the user. On the *Flickr* and *LastFM* experiments, we have computed the *variance* which enables to compute the variability of the recall and is computed as follows: $V(X) = 1/N \times \sum_{i=1}^{n}(x_i - m)^2$ where m is the average recall.

In our experiments, we use the following relevance scores:

$\mathbf{overlap}(\mathbf{u}, \mathbf{v}) = |I_u \cap I_v|$ $\quad$ $\mathbf{over\_big}(\mathbf{u}, \mathbf{v}) = |I_u \cap I_v| + |I_v|$

$\mathbf{Jaccard}(\mathbf{u}, \mathbf{v}) = |I_u \cap I_v| \ / \ |I_u \cup I_v|$ $\quad$ $\mathbf{cosine}(\mathbf{u}, \mathbf{v}) = Ir_u \times Ir_v \ / \ ||Ir_u|| \times ||Ir_v||$

where I_u and I_v are the items shared by u and v, respectively, and where Ir_u and Ir_v are the set of ratings u and v gave to the items they share. We have fixed the *U-Net*'s size to 16 and *TTL* to 3. Other values have been tested and showed similar results. The size of the random view (5 in our case) is not important as it only modifies the convergence speed.

4.2 Experiments

Figure 2 presents the results of our experiments. More precisely, Figures 2a, 2b and 2c compare the recall results of the used relevance scores with and without including our usefulness score, while Figures 2d, 2e and 2f compare the recall results of several diversification methods.

[1] `www.peersim.sourceforge.net`

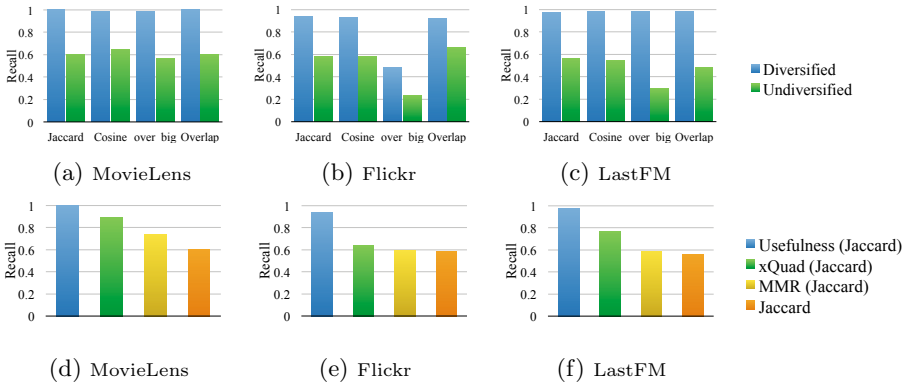


Fig. 2. Effect on recall of diversification

Not surprisingly, diversifying the *U-Net* enables for all relevance score to significantly increase recall. On the *MovieLens* dataset, the recall results without diversification range between 0.58 and 0.62 while they range between 0.978 and 0.999 with diversification. On the *Flickr* dataset, the gains are slightly smaller. Since all users share their own pictures, their profiles are very different and already diversified. Therefore, diversification has less impact on the recall. Finally, the *LastFM* dataset recall results are up to 3.26 times higher.

In addition to improve the recall, diversified solutions enable to reduce the variance compared to undiversified solutions. For instance, on *Flickr*, the variance decreases from 0.116 to 0.013 when using `Jaccard`. This can be explained by the fact that in the undiversified solution, users in *U-Net* are very similar among them. As a consequence, either all are relevant to the query, and hence they provide a high recall; or none of them is, thus producing a low recall. Diversification enables to increase coverage and therefore, it increases the probability to answer any kind of query.

In addition, we ran these experiments with different sizes of *U-Net* and values of *TTL*. For instance, on the *MovieLens* dataset, with a *U-Net* of size 5 and a *TTL* of 2, the recall is in average 2.37 times higher compared to undiversified solutions. Indeed, without diversification, recall values are in average of 0.26 while they reach $0,61$ using usefulness.

We have also compared three different diversification methods. The first is the *usefulness* score presented in Equation 7. The second method we use is the *Maximal Marginal Relevance*, known as *MMR* [13]. *MMR* chooses users that minimize the maximum similarity between any two users in u's *U-Net*. Finally, the last method is *Explicit Query Aspect Diversification* known as *xQuad* [14]. *xQuad* chooses users such that each user v_i in u's *U-Net* is similar to u in a different way. For instance, suppose that u shares items i_1 and i_2. If v_1 is in u's *U-Net* and is similar to u because it also shares i_1, then, *xQuad* chooses a user v_2 such that v_2 is similar to u because it shares the item i_2. In this experiment, we use *Jaccard* as the similarity measure.

Figures 2d, 2e and 2f show that all diversification methods enable to increase the recall values compared to undiversified methods. Among them, *usefulness* obtains the best gain in terms of recall closely followed by *xQuad*. Finally, *MMR* shows the worst gain in terms of recall. Indeed, *MMR* chooses users that minimize the maximum similarity between any two users in u's *U-Net*. Therefore, it prefers users that are a little bit similar with every user in u's *U-Net*, and that do not necessarily increase recall results.

5 Related Work

Distributed recommendation for web data based on collaborative filtering has been recently proposed with promising results. In this section, we compare our recommendation approach with state of the art solutions.

In [15], Loupasakis and Ntarmos propose a decentralized approach for social networking with three goals in mind: privacy, scalability with profitability and availability. They propose an architecture based on a DHT for keywords query search. Since, DHTs are better suited for exact-match queries, the author propose to decompose each query into several single word exact-match queries. The main drawback is that responses that have medium scores with respect to each keyword but high scores with respect to all the keywords are likely to be missed.

P2PRec [3] is a gossip-based search and recommendation solution where the profile of each user u is represented as a set of topics computed based u"s items. Then, using gossip protocols, similar users in terms of topics, are clustered together and used to guide recommendation as we do. However, since diversity is not taken into account, users within each cluster can be redundant, thus limiting recall results. In [6], Kermarrec et al. focus on recommendation and propose to combine gossip algorithms and random walks. The users are clustered based on relevance through gossip protocols. A user has knowledge of the items shared by its neighbors. To compute the recommendation, each user runs locally a random walk using a transition similarity matrix. However, the computational complexity of the algorithm with respect to the size of the neighborhood and the number of items. reduces the complexity of the approach. Moreover, Kermarrec et al. [7] claim that, since users are heterogeneous, the similarity measure used to cluster users should also be heterogeneous. Nevertheless, the concept of diversity is different from ours as it represents the usage of various relevance scores depending on each user"s profile. As a consequence, each user"s cluster may still carry redundant user profiles, because there is no explicit diversification. In [1], Bai et al. propose a solution for personalized *P2P top-k search* in the context of collaborative tagging systems, called *P4Q*. In this solution, the users are clustered based on relevance through gossip protocols. The users in each cluster are split into two groups: 1) the c closest users from which u replicates all items metadata (*i.e.* tagging actions) and 2) the n less similar users from which u knows only the profile. Still, diversity is not taken into account and users within the clusters are likely to be redundant.

6 Conclusion and Future Work

In this paper, we proposed a new gossip-based search and recommendation approach with new measures and techniques. We first showed that usefulness, by combining relevance and diversity, is very effective in increasing recall results and can be used as a clustering score. Then, we designed a new clustering algorithm based on usefulness that combines relevance and diversity. We validated our proposal with an experimental evaluation using several datasets and show major gains with recall results more than two times better.

In future work we intend to exploit other recommendation scenarios such as multisite recommendation.

References

1. Bai, X., et al.: Collaborative personalized top-k processing. Transactions on Database Systems 36(26) (December 2011)
2. Carretero, J., et al.: Geology: Modular georecommendation in gossip-based social networks. In: ICDCS, pp. 637–646 (2012)
3. Draidi, F., Pacitti, E., Parigot, D., Verger, G.: P2Prec: a social-based P2P recommendation system. In: CIKM, pp. 2593–2596 (2011)
4. Voulgaris, S., van Steen, M.: Epidemic-style management of semantic overlays for content-based searching. In: Cunha, J.C., Medeiros, P.D. (eds.) Euro-Par 2005. LNCS, vol. 3648, pp. 1143–1152. Springer, Heidelberg (2005)
5. Jelasity, M., Babaoglu, O.: T-man: Gossip-based overlay topology management. In: Brueckner, S.A., Di Marzo Serugendo, G., Hales, D., Zambonelli, F. (eds.) ESOA 2005. LNCS (LNAI), vol. 3910, pp. 1–15. Springer, Heidelberg (2006)
6. Kermarrec, A., Leroy, V., Moin, A., Thraves, C.: Application of random walks to decentralized recommender systems. In: Lu, C., Masuzawa, T., Mosbah, M. (eds.) OPODIS 2010. LNCS, vol. 6490, pp. 48–63. Springer, Heidelberg (2010)
7. Kermarrec, A., Taïani, F.: Diverging towards the common good: Heterogeneous self-organisation in decentralised recommenders. In: SNS, pp. 3–8 (2012)
8. Angel, A., Koudas, N.: Efficient diversity-aware search. In: SIGMOD, pp. 781–792 (2011)
9. Chen, H., Karger, D.: Less is more: Probabilistic models for retrieving fewer relevant documents. In: SIGIR, pp. 429–436 (2006)
10. Manning, C., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press (2008)
11. Kowalczyk, W., Jelasity, M., Eiben, A.: Towards data mining in large and fully distributed peer-to-peer overlay networks. In: BNAIC, pp. 203–210 (2003)
12. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison-Wesley Longman Publishing Co., Inc. (1999)
13. Carbonell, J., Goldstein, J.: The use of MMR, diversity-based reranking for reordering documents and producing summaries. In: SIGIR, pp. 335–336 (1998)
14. Santos, R., Peng, J., Macdonald, C., Ounis, I.: Explicit search result diversification through sub-queries. In: Gurrin, C., He, Y., Kazai, G., Kruschwitz, U., Little, S., Roelleke, T., Rüger, S., van Rijsbergen, K. (eds.) ECIR 2010. LNCS, vol. 5993, pp. 87–99. Springer, Heidelberg (2010)
15. Loupasakis, A., Ntarmos, N.: eXO: Decentralized autonomous scalable social networking. In: CIDR, pp. 85–95 (2011)

Towards Privacy for MapReduce on Hybrid Clouds Using Information Dispersal Algorithm*

Asma Ben Cheikh[1], Heithem Abbes[1], and Gilles Fedak[2]

[1] Université de Tunis, LaTICE, ENSIT, 5 Avenue Taha Hussein, Tunis, Tunisie
{asma.bencheik,heithem.abbes}@latice.rnu.tn
[2] Université de Lyon, INRIA, France
gilles.fedak@inria.fr

Abstract. MapReduce is a powerful model for parallel data processing. The motivation of this work is to allow running map-reduce jobs partially on untrusted infrastructures, such as public clouds and desktop grid, while using a trusted infrastructure, such as private cloud, to ensure that no outsider could get the 'entire' information. Our idea is to break data into meaningless chunks and spread them on a combination of public and private clouds so that the compromise would not allow the attacker to reconstruct the whole data-set. To realize this, we use the Information Dispersion Algorithms (IDA), which allows to split a file into pieces so that, by carefully dispersing the pieces, there is no method for a single node to reconstruct the data if it cannot collaborate with other nodes. We propose a protocol that allows MapReduce computing nodes to exchange the data and perform IDA-aware MapReduce computation. We conduct experiments on the Grid'5000 testbed and report on performance evaluation of the prototype.

1 Introduction

MapReduce is a powerful parallel data processing model, capable of simplify the programing of data-intensive applications, i.e applications manipulating an enormous amount of data at large scale. Recently, many organizations have adopted the MapReduce model and have implemented their own frameworks such as Google MapReduce [DG04], Yahoo! Hadoop [Whi09] and BitDew [TMC+10]. Furthermore, this model has been adopted by many researchers in high performance computing, data intensive scientific analysis, large scale semantic annotation and machine learning.

A large class of data processing systems using MapReduce is mainly running on local platform, such as clusters. However, open systems such as Service Oriented Architecture, Grid Computing, Volunteer Computing and Cloud Computing offer platforms to run MapReduce applications on. In particular, Cloud

* This work is supported by the French Agence Nationale de la Recherche through the MapReduce grant under contract ANR-10-SEGI- 001-01, as well as INRIA ARC BitDew.

A. Hameurlain et al. (Eds.): Globe 2014, LNCS 8648, pp. 37–48, 2014.

Computing aims to offer affordable and scalable computing capacities, which meet the needs of MapReduce applications. However, because of the lack of security mechanisms to ensure data privacy provided by Cloud providers, users are still reluctant to offload the processing of their sensible data-sets.

Desktop Grids (DG) [CF12] are a form of volunteer computing that have known success thanks to the high computing and storage power they offer with a low economic cost. The architecture of this infrastructure is based on the federation of free resources; users, voluntarily, participate with their machines when these are idle. Volatility and security are ones of the constraints that discourage users to exploit this enormous potential.

Our contribution is to enhance MapReduce security, so that it protects data sent by the users to remote computing infrastructures from leakage and eavesdropping. More specifically users face two kinds of threats : 1) during data distribution, an eavesdropper could intercept data when being transferred, and 2) when stored or processed, a malicious workers could have access to data. Unfortunately, if encryption can protect data transfer and storage, it cannot prevent the spying of data when they are deciphered for computation. There exists techniques that allow to process encrypted data, however, those are not yet generic enough for supporting any kind of computation.

As MapReduce is based on parallel processing, data has to be divided over the computing nodes so each one processes a chunk as an input file. To improve data privacy, our approach is to use a combination of trusted and untrusted infrastructures, for instance private and public Clouds, to store the data set and execute the MapReduce applications. Our approach relies on the Information Dispersal Algorithm (IDA) to split and distribute the data.

Our idea is to break data into meaningless chunks so that a malicious worker or eavesdropper, can not get access to meaningful data. A meaningless data is an obsolete and useless information so even if a malicious worker has access to it, the data (i.e the meaningfull) remains protected. To do so, we use IDA which generates, from an input file, several chunks and disperses them on several machines. Each machine aiming to access data has to contact other machines to get missing chunks to reconstruct the needed information. In our case, we call chunk provided by IDA: meaningless data. So, if a malicious node has 1 chunk, it has to contact and collaborate with other nodes to get missing ones. The lack of one chunk prevents the malicious user to get access to meaningful data. In order to hide some chunks from malicious users, we use a hybrid cloud infrastructure. If m chunks are necessary to reconstruct data, we deploy $m - 1$ chunks on untrusted infrastructure, such as public cloud and desktop grid. The remaining chunks are deployed on a private cloud. We assume that a private cloud is highly secure and cannot be accessible by malicious users.

The rest of the paper is organized as follows. Section 2 presents the dispersion algorithm IDA and MapReduce. In Section 3, we describe our approach with its various components. Section 4 analyzes the experiments results. Section 5 exposes related works. Finally, we conclude in section 6.

2 Background

2.1 Presentation of IDA

Information Dispersal Algorithm (IDA), first proposed by O. Rabin in 1989 [Rab89], is used to break a file F of length $L = |F|$ into n pieces F_i, i in $[1..n]$, each of length $|F_i| = L/m$, so that every m pieces suffice to rebuild F. The sum of the lengths F_i is $(n/m) * L$. With IDA, since any m pieces, among the n created, can reconstruct the data, loosing parts of data when stocking or routing can be remedied.

Mainly, IDA is based on two key parameters. As defined, n is the total number of pieces resulting from IDA, of which any m chunks are sufficient to reconstruct the original file. m is called "quorum" or "threshold". O. Rabin defines the ratio between m and n as follows: $n/m = 1 + e$ $(e > 0)$, so it ensures greater reliability of the algorithm in terms of security and storage capacity.

IDA is based on two routines. The *split* operation generates the n chunks to be distributed. The *combine* routine requires m chunks passed as arguments to reconstruct the original data.

Both algorithms of division (split) and recomposition (combines) operate by performing a multiplication of the entry and a key matrix, also called transformation matrix.

The key matrix must have the property that every subset of m different vectors are linearly independent. If this is not the case, the key matrix can not be inverted and the combine phase is not feasible. Indeed, the key matrix is generated from a "key". It is defined as a vector V_{key} composed of a list of elements respecting the following constraints:

$$V_{key} : (x_1, ..., x_n, y_1, ..., y_m) \text{ , for every } i, j :$$
$$(1)\ x_i + y_j \neq 0 \ ; \ (2)\ i \neq j \rightarrow x_i \neq x_j \text{ et } y_i \neq y_j$$

A row of the key matrix is:

$$a_i = (\tfrac{1}{x_i + y_1}, ..., \tfrac{1}{x_i + y_m})$$

First, The key matrix $A(n * m)$ is generated respecting the mathematical constraints seen above. The next step is to decompose the input data into sequences. Each sequence Si is expressed as a vector F_i. F_i is composed of m elements. An element is of size w and can be a character $(w = 1)$, a word, etc.. We will call the matrix composed of the different vectors F_i, the matrix F. Then, F is transposed to have F_t. The multiplication of A by F_t generates a matrix R (IDA output). A chunk M_i is a file composed of 1) a header containing the key vector of order a_i (i.e. the i^{th} row of the key matrix), 2) a body storing R_i. M_i is a *chunk* to send to a mapper.

Once m chunks are collected by a host, the key matrix is obtained from the m headers. The product result of the inverted key matrix by the content of the chunks gives the original data.

2.2 MapReduce

The principle of MapReduce consists of splitting the data into parts so their process is simultaneously done. The model defines two main functions, the Map function processes input chunks and the Reduce function processes the output of Map tasks and outputs final results.

Computing elements can be classified into Mapper nodes, which execute Map tasks and Reducers which execute Reduce tasks. In a first step, input data are divided into chunks and distributed over the Mappers. Then, Mappers apply the Map function on each chunk. The result of the execution of a Map task is list(k,v), a list of key and value pairs. An intermediate Shuffle phase sorts the map outputs, called intermediate results, according to keys so that in a second step, each Reducer processes a set of the keys. In the Reduce phase, Reducers apply the Reduce function to all of the values (k, list(v)) for a specific key. At the end, all the results can be assembled and sent back to the user.

3 Our Secure MapReduce Approach Based on IDA

Our approach is composed of four phases, illustrated in Figure 1 as follows:

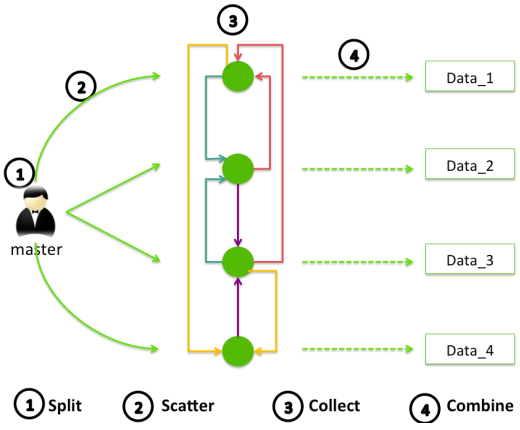


Fig. 1. IDA Phases for MapReduce

3.1 Preparation Phase (Split)

Classical MapReduce master splits the input data into chunks based on the chunk size. In our approach, we apply the IDA split routine on input data which generates n chunks. A chunk M_i is composed of 1) a header containing the key vector of order i (i.e. the i^{th} row of the key matrix), 2) a body storing the i^{th} row of the key matrix and data product. All generated chunks will be dispersed to the mappers machines.

3.2 Distribution Phase (Scatter)

Once we get n chunks by applying the split routine, the master sends them to the mapper machines via messages called *yourChunk*() (see Figure 2).

At this point, no mapper can start executing its task because the data it owns do not match the real input data, because IDA applies multiplications that alters data without losing them. Therefore, a mapper should contact other machines to have the necessary information allowing the extraction of a sequence of meaningful data during the collect phase.

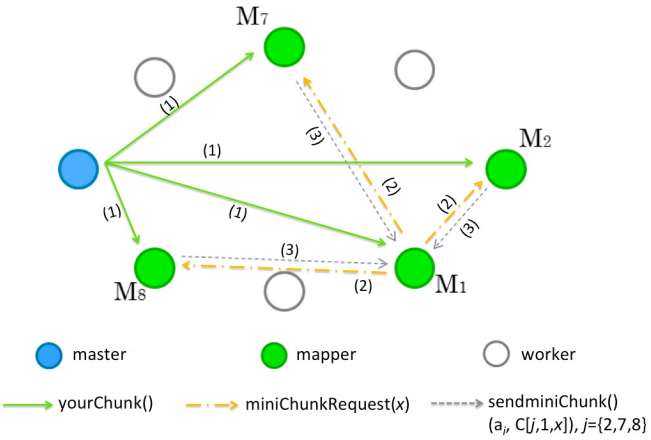


Fig. 2. Exchanged data and messages

3.3 Collect Phase

Each mapper randomly selects its friends among all the mappers who received their chunks. A mapper list of friends must be composed of $m - 1$. Adding to them its data, m is the number of keys necessary for calculating the inverse matrix. Then, it contacts them to ask for their *miniChunks*.

The request is made by sending a *miniChunkRequest*() message. Asking an element from each friend allow the mapper to restore only one sequence which size is m. IDA specifies m as the size of a sequence, which is a very fine granularity and may degrade the performance of our system. To solve this problem, each mapper is responsible of restoring a serie of contiguous sequences that we call packet. Its size is fixed by the master. To do so, a mapper specifies in the request message sent to its friends, the order of packet it should restore. Each friend generates, from its own chunk, the appropriate mini chunk corresponding to the order of requested packet and its size. Then, as an answer to the miniChunkRequest(), they send them a *sendminiChunk*() message (see Figure 2).

If the mapper Map1 is responsible for processing the packet number 1 or $P1$, it sends a message $miniChunkRequest(1, pack\_size)$ to $m-1$ mappers (assume $m = 4$). Each mapper Map_j (j in $[2..n]$) answers with a message $SendminiChunk(a_j, R[j, 1, x])$, where a_j is a header of the chunk j that is indeed the j^{th} vector of the key matrix A and $R[j, 1, x]$ is the vector composed of x elements starting from the first element in the chunk received by the Map_j mapper (see Figure 2).

In classic MapReduce frameworks, there is no communication between mappers.

3.4 Extraction Phase (Combine)

Once the data is collected, each mapper first determines the inverse key matrix from the m keys that it possesses (its own and those it had just received in the collect phase). Thereafter, by multiplying the inverse matrix by the vector composed of the elements collected, each mapper gets the packet to perform the map task on it. These instructions match the treatments performed by the IDA Combine function. Figure 3 summarizes the steps of this phase carried out by the mapper Map1.

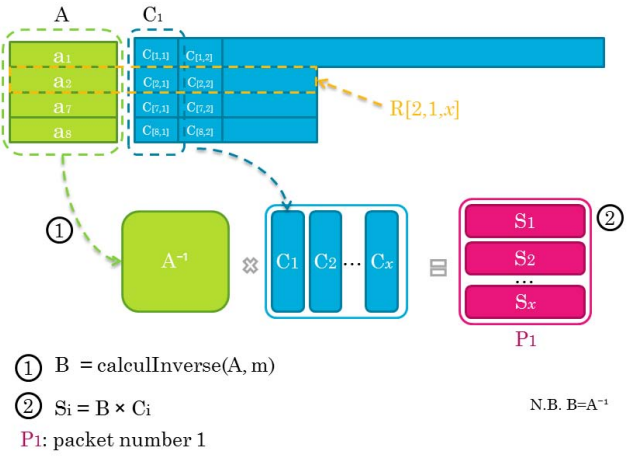


Fig. 3. Extraction of data by Map1

Comparing our approach to classical MapReduce systems, the preparation and scatter phases are the same. Our preparation phase may take longer time, because it does not only split data but apply a more complex procedure. Then classically, workers start immediately their map tasks, while in our approach, each mapper needs extra treatment to clarify its data. The procedure is close to deciphering with a shared key. Mapper must collect the key to decrypt its data in the extraction phase.

3.5 Requirements

- **Ratio between n and m:** As m is the number of chunks necessary and sufficient to reconstruct the input file, it is advantageous to maximize it. On the other hand, this ratio ensures that the size of manipulated data is close to the original file size, as extracting n chunks of length $|F|/m$ each, from a file size $|F|$ gives a percentage of $((n/m * |F|) - |F|) * 100/|F|$ of redundancy. Otherwise, with $n = 2m$, we will double (100%) the source file to generate the n files to scatter.

  In our approach, $2n*(m-1)$ messages are exchanged. Therefore, the choice of parameters n and m has impact on the performance of our approach.
  - if $m \ll n$: we reduce communications but we weaken the security and considerably increase redundancy.
  - if $m \approx n$: we provide better security, we maintain an acceptable level of redundancy but we increase communications between mappers.
- **Mappers allocation:** Threats can occur at the mapper itself, as a malicious one, or during communications when intruders hearken the network. A malicious mapper can have access to data as it is charged to process it. This scenario is allowed in our system. Nevertheless, when a community of, at least, m malicious mappers cooperate to reveal their input, they may reveal all data, not only theirs. To prevent that, we propose to use a hybrid cloud infrastructure to deploy our solution. On each public cloud, we deploy $m-1$ chunks.This number of chunks is not sufficient to reconstruct all the data. The remaining chunks will be deployed on a private cloud.

  There are different scenarios when taking into account the existing Cloud providers, their cost and, the most importantly, confidence and probable threats that may occur to each. A first given scenario may divide data between two famous Cloud such Amazon and IBM because security techniques are more reliable, and the cost would be relatively higher. A second scenario would choose others less trusted Clouds, so that first the cost is lower, second, the user may allow a given level of data visibility; i.e the number of eventual untrusted mappers. The user could choose between different scenarios according to his application and his data requirements.

4 Experiments and Evaluation

We have implemented our approach in Perl, to manage communication between mappers and reducers and we have used Crypt-IDA[1] library, which is an implementation of IDA in perl.

We realized a set of experiments on the Grid'5000 platform using 220 machines on the Nancy site.

In order to evaluate the performance of our system, we chose to evaluate the phases according to their locality of execution, the first two phases, Split and Scatter (step 2S), being executed by the master, the two last Collect and

[1] `http://search.cpan.org/~dmalone/Crypt-IDA-0.01/lib/Crypt/IDA.pm`

Combine (step 2C) performed on each mapper. Indeed, we have made different measurement scenarios in order to study the impact of various parameters on the execution time on each phase on the one hand. On the other hand, the various measures allow us to calculate the overhead (data overload) resulting in Split phase and its impact on the Scatter phase. Finally, the analysis of experiments guide us to the optimal choice of the parameters of each scenario.

4.1 Evaluation of the Split and Scatter Step (2S)

The first experiment aims to study the impact of input data size on the execution time of Split Phase. We have turned the split routine on 8 cores node by varying the size of the input file from 100MB to 1.3GB, and setting the parameters n=25 and m=10. As a result, the overhead rate is 150%. Figure 4 shows the execution time in function of the input file size . As we can see, the split time increases with the size of the input data. This is explained by the fact that applying the split routine on a larger file is to multiply by a matrix of larger size and generate larger pieces.

We noted that the preparation phase of chunks does not produce a significant time overhead compared with MapReduce systems when dividing data into chunks to send to mappers.

A second experiment aims to study the impact of n, the number of chunks generated from split, on the execution time of the split and scatter phases. We applied the split routine on a 1.3 GB file size, by changing n between 25, 82 and 180. Note that the results shown in Figure 5 show the average of values found by varying the parameter m; for n = 25, m varies from 10 to 24, for n=82, m=35,40,45,50,55,60 for n=180, m=68,69,70. The results show that for the same size of input data, the duration of the split phase increases with n. Such is also the case of the scatter Phase. The larger n is, the more key matrix, involved in multiplication, increases in size. We deduce that it is better to maximize the number of pieces to optimize the execution time.

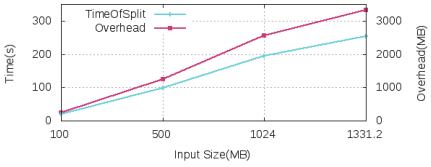


Fig. 4. Duration of Split in fuction of input data size

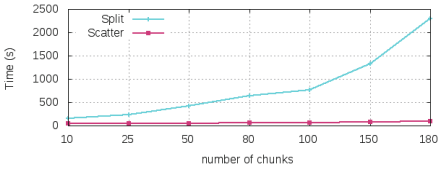


Fig. 5. Impact of n on execution time

4.2 Evaluation of the Collect and Combine Step (2C)

In order to start the map task, each mapper needs to restore the package of meaningful data. To do so, it must contact m-1 friends-mappers, of which it

requests the information necessary to rebuild the package that has been assigned. Two parameters are involved, the number of friends: m, and the size of data to be restored: the packet size.

In order to study the impact of the packet size to generate over step 2C, we performed a first experiment which calculates the execution time depending on the packet size, which varies from 5 to 128 MB. During the execution of the step 2C, the number of chunks generated in the split phase is not important. What matters in the implementation stage is the number of available mappers. A maximum number of mappers ensures that the 2C step takes place in a minimum number of iterations, a single iteration if possible. With 1.3GB as input data size and 5MB as packet size, we would have to treat 260 packets, against only 11 for a packet size of 128MB. The execution time is composed of Collect phase and Combine phase.

Figure 6 presents the results found. As we can see, a packet size equal to 5MB generates a duration of the Collect phase up to the double for other sizes where the duration does not change practically. Nevertheless, we can see a minimum time with a packet size between 10 and 16MB. Indeed, the duration made by the system with a 5MB packet includes two iterations taken to process the 260 packets affected to 180 mappers. As for the duration of the combine phase, we reach a minimum , also with packet size between 10 and 16MB. For the rest, the more the packet size is, the more combine duration rises. This increase is explained by the fact that applying the rabin-combine routine on larger data, returns to handle larger matrices.

We can conclude that the duration of a single iteration 2C is minimal with a smaller packet size. In contrast, a small size generates more packets. Which leads to more iterations for assigning all the packets. Accordingly the right choice is to provide an appropriate size, according to the number of available mappers n, to minimize the total number of iterations. In our example, to be closer to the optimal value of the systems implementing the MapRdeuce model, we choose 16 MB for the rest of experiments.

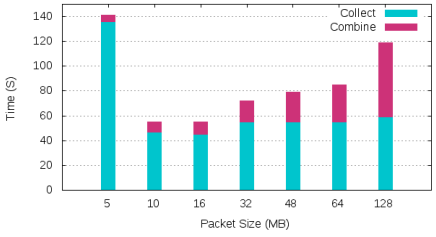


Fig. 6. Time of Collect & Combine according to the packet size

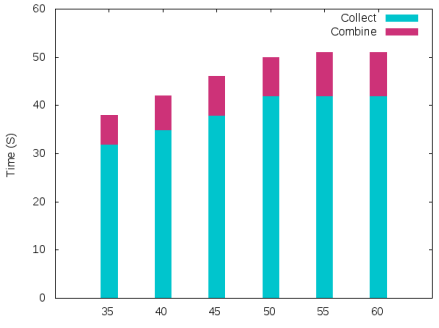


Fig. 7. Time of Collect^{m} & Combine according to m

A second experiment questions the choice of m, which is the number of friends to contact during collect phase and the number of input files for the combine routine. The experiment consists in measuring the execution time of the two phases in terms of m. The data size to disperse is 1.3GB. The packet size is 16MB. These values generate 82 packets. To proceed in a single iteration, we set n=82. Therefore, m is from 35 to 60. Time includes the duration of the two phases: Collect and Combine. Figure 7 illustrates the different results found. We can observe that more m is small, we get better results.

In consideration, evaluating the step 2S and more specifically the split phase, we were brought back to deduce that m gives better execution time when getting closer to n, exceeding n/2. To decide about the choice of m, we re-ran the previous experiment. Figure 8 includes all the phases; execution times of all phases in terms of m. We observe that the value of m which gave a minimum time in step 2C, does not lead to a global optimum time. We conclude that the optimal value of m is n/2.

4.3 Scenarii and Discussion

Based on the results obtained in experiments questioning the impact of the number of chunks produced and the size of the packets, we are in front of two alternatives. On the one hand, we discovered that our system takes longer to generate more chunks (Phase Split). On the other hand, we found that the optimal packet size leading to a minimum execution time is 16MB. For data of 1,3GB, we obtain 83 packets to be handled. Would it better to generate 83 chunks and assign each packet to a mapper so that step 2C (Collect and Combine) is carried out in a single iteration? Or would it be more appropriate and efficient to generate less chunks to save time during the split phase, and accept that the step 2C occurs in more than one iteration?

To answer these questions, we performed a comparative study between the two scenarios. We realized four versions measuring each time the length of four phases. In each experiment, we varied the number of mappers from 25 to 83. Figure 9 shows the overall execution time. The first experiment done with 25 mappers takes bit of time to create chunks, while it generates four iterations of the step 2C. 40 mappers require 40 chunks, which increases the duration of the split phase compared with the first experiment, but treat all packets in 3 iterations. 50 mappers require more time to generate their corresponding chunks, but complete the step 2C in two iterations. As for 83 mappers, the master puts more than 600 seconds to perform the split routine and prepare the chunks. And although the two phases Collect and Combine ending in a single iteration of assignment, the overall time presents a significant additional cost compared to other senarios. The total overcost is due, primarily and directly, to the split phase run locally on the master.

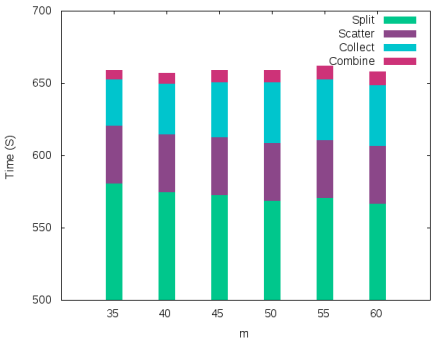


Fig. 8. Total time according to m

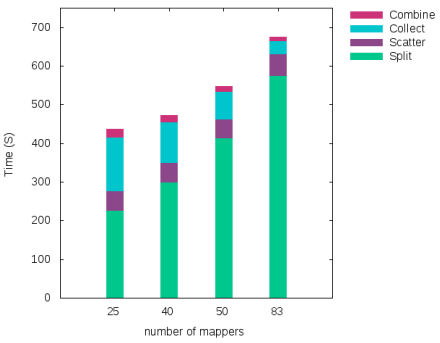


Fig. 9. Comparative study

5 Related Works

The problem we face in this work is protecting MapReduce applications data in public environments. Existing solutions focus on data when they have been sent to be processed; either they apply control techniques such as Mandatory Access Control (MAC) [Abr90, McC04] , or results verification and results control techniques [Dwo10, Dwo11] which have been used by systems like Airavat [RSK+10] and SecureMR [WDYG09] to ensure security, integrity and privacy for MapReduce during running the application. None of these systems has considered the threats that may occur during the dispersal of data over the working machines deployed on public clouds or desktop grids.

IDA has been essentially exploited for file sharing systems [Rab90, DFM00]. It was applied to provide a secure and reliable storage of information, and supply fault-tolerant and efficient transmission of information in networks. The concern was how to prevent loss of data when stocking without having to duplicate data and provide enormous capacities, or when transmitting avoiding sending multiple copies and charging the network. With IDA, since any m pieces, among the n created, can reconstruct the data, loosing parts of data when stocking or routing can be remedied.

6 Conclusion

We have proposed a new approach of securing data distribution for MapReduce applications, using Information Dispersal Algorithm. IDA is a mechanism that allows to split a file into pieces so that, by carefully dispersing the pieces, there is no method for a single node to reconstruct the data unless it cooperates with others. We have implemented a prototype that adapt IDA to MapReduce needs while respecting privacy constraints. We have realized several experiments to evaluate our prototype performance.

In a future work, we plan to integrate our protocol in distributed MapReduce frameworks such as hadoop and BitDew.

References

[Abr90] Eggers, K.W., LaPadula, L.J., Olson, I.M., Abrams, M.D.: A generalized framework for access control: an informal description. In: Proceedings of the 13th NIST-NCSC National Computer Security Conference, pp. 135–143 (1990)

[CF12] Christophe Cérin and Gilles Fedak. Desktop grid Computing. CRC Press (June 2012)

[DFM00] Dingledine, R., Freedman, M.J., Molnar, D.: The free haven project: Distributed anonymous storage service. In: Federrath, H. (ed.) Designing Privacy Enhancing Technologies. LNCS, vol. 2009, pp. 67–95. Springer, Heidelberg (2001)

[DG04] Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. In: Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation, OSDI 2004, vol. 6, p. 10. USENIX Association, Berkeley (2004)

[Dwo10] Dwork, C.: Differential privacy in new settings. In: Charikar, M. (ed.) Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 174–183. SIAM (2010)

[Dwo11] Dwork, C.: Differential privacy. In: Encyclopedia of Cryptography and Security, 2nd edn., pp. 338–340 (2011)

[McC04] McCarty, B.: Selinux: Nsa's open source security enhanced linux. O'Reilly and Associates (2004)

[Rab89] Rabin, M.O.: Efficient dispersal of information for security, load balancing, and fault tolerance. J. ACM 36(2), 335–348 (1989)

[Rab90] Rabin, M.O.: The information dispersal algorithm and its applications. In: Capocelli, R.M. (ed.) Sequences, pp. 406–419. Springer, New York (1990)

[RSK+10] Roy, I., Setty, S.T.V., Kilzer, A., Shmatikov, V., Witchel, E.: Airavat: security and privacy for mapreduce. In: Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, NSDI 2010, p. 20. USENIX Association, Berkeley (2010)

[TMC+10] Tang, B., Moca, M., Chevalier, S., He, H., Fedak, G.: Towards mapreduce for desktop grid computing. In: Proceedings of the 2010 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2010, pp. 193–200. IEEE Computer Society, Washington, DC (2010)

[WDYG09] Wei, W., Du, J., Yu, T., Gu, X.: Securemr: A service integrity assurance framework for mapreduce. In: Proceedings of the 2009 Annual Computer Security Applications Conference, ACSAC 2009, pp. 73–82. IEEE Computer Society, Washington, DC (2009)

[Whi09] White, T.: Hadoop: The Definitive Guide. Definitive Guide Series. O'Reilly Media, Incorporated (2009)

An Elastic Approximate Similarity Search in Very Large Datasets with MapReduce

Trong Nhan Phan[1], Josef Küng[1], and Tran Khanh Dang[2]

[1] FAW Institute, Johannes Kepler University Linz, Austria
{nphan,jkueng}@faw.jku.at
[2] HCMC University of Technology, Ho Chi Minh City, Vietnam
khanh@cse.hcmut.edu.vn

Abstract. The outbreak of data brings an era of big data and more challenges than ever before to traditional similarity search which has been spread to a wide range of applications. Furthermore, an unprecedented scale of data being processed may be infeasible or may lead to the paralysis of systems due to the slow performance and high overheads. Dealing with such an unstoppable data growth paves the way not only to similarity search consolidates but also to new trends of data-intensive applications. Aiming at scalability, we propose an elastic approximate similarity search that efficiently works in very large datasets. Moreover, our proposed scheme effectively adapts itself to the well-known similarity searches with pairwise documents, pivot document, range query, and k-nearest neighbour query. Last but not least, these methods, together with our filtering strategies, are implemented and verified by experiments on real large data collections in Hadoop showing their promising effectiveness and efficiency.

Keywords: Similarity search, approximate search, very large datasets, MapReduce, Jaccard measure, Hadoop.

1 Introduction

Similarity search has been widely used and played an essential role supporting a variety of applications. The basic goal is to find possibly relevant results whenever a query sent by a user. In fact, many application scenarios want to retrieve objects that look like a given one or find the most similar objects in databases. Some typical instances include plagiarism detection, recommendation systems, and data cleaning or clustering in data mining, to name a few. Nevertheless, similarity search has encountered a barrier caused by a big volume of data. These large datasets, from web crawlers, data logs, or click streams for example, may be processed up to hundreds of terabytes of data. In order to identify how similar a pair is, quantitative methods known as similarity functions, metrics, or distance measures are applied. It also means computation costs are additionally added to derive similarity scores. In other words, if we have n objects, the cost for pairwise similarity computing is $O(n^2)$. Such a high cost would be a burden to the system when objects never stop quickly growing, or they are distributed over a big number of computing nodes.

A. Hameurlain et al. (Eds.): Globe 2014, LNCS 8648, pp. 49–60, 2014.

Researchers not only from academia but also from industry have focused on how to tackle the issue for many years. Some try to propose novel indexes or ways to approximately but efficiently do similarity search [5][6]. Recent state-of-the-arts pay attention to parallelism by optimizing algorithms in parallel [1] or employing MapReduce [9] to improve similarity search for large scale data [4][8][12][13][14]. In this paper, we propose an elastic scheme dealing with massive datasets. Our main contributions are as follows:

1. A general approximate similarity search scheme with MapReduce is proposed towards data scalability.
2. Collaborative refinement strategies are exploited to meet users' search needs and reduce candidate size, which leads to eliminating unnecessary similarity computing and turning storage overheads down.
3. The scheme is then shown how much adaptable it is to specific similarity search including pairwise documents search, pivot document search, range search, and k-Nearest Neighbor (k-NN) search.
4. Experiments are conducted with Apache Hadoop framework [3] and real large datasets to indicate which can benefit from the proposed methods when large amounts of data keep increasing.

The rest of paper is organized as follows. Section 2 introduces relevant researches in the field of similarity search which are close to our work. Section 3 presents some preliminaries behind our proposed similarity search methods. Next, section 4 shows our general scheme with MapReduce while section 5 discusses about how the general scheme adapts to the four most common similarity search strategies. Section 6 gives our experiments before we conclude our work and its future in section 7.

2 Related Work

The importance of similarity search as well as its issues has gained much attention and effort from researchers world-wide, especially when the data become oversized. While the traditional exact similarity search demands full computations that has the complexity as $O(n^2)$, much work tries to reduce the search space by filtering methods or approximately achieves similarity to improve the performance. In [11], the authors propose State Set Index, which is interpreted as a nondeterministic finite automaton, as a method for fast similarity search in very large string sets. Another approach coming from the work in [15] utilizes positional filtering principle which is then combined with prefix and suffix filtering. The work in [16] presents the combination between index structure based method for prune strategy and hashing based method for fast distance computation. Nevertheless, only k-NN query is considered. In general, these methods are sequentially done without caring about parallel mechanism.

Still there is other work exploiting parallelism to do similarity search. Like in [1], the authors conduct exact all-pair similarity search. Nevertheless, it requires a static partitioning to assure that dissimilar vectors are assigned to different partitions before computing similarity pairs. The authors in [10] introduce a MapReduce algorithm to compute the similarity score between pairs in large document collections. The algorithm consists of two MapReduce phases which first create an inverted index and then

take pairwise similarity into account. Nevertheless, the algorithm only considers term frequency between documents as weights accumulating to the final similarity score by their inner product. This way does not bring much accuracy to the final score measuring similarity. Furthermore, the authors do not mention any methods to reduce the size of candidate pairs. That means each document in the large collections is compared one by one, which brings quadratic computing costs when the size of data rapidly increases. From the work in [12], the authors also use MapReduce as the skeleton of their approach. The costs for MapReduce operations, however, are so high due to so many phases. There are two MapReduce jobs to construct the dictionary, one MapReduce job to transform texts into vector texts, one MapReduce job to calculate the inverted file, and two MapReduce jobs to resolve the query text. Moreover, the big size of the prefixes will cause computing overheads and slow down the system when massive datasets are given. The authors in [4] propose 2-phase MapReduce for the self-join problem. The first MapReduce job is to build an inverted index whereas the second MapReduce job is to calculate similarity scores based on partial contributions. Their approach requires overriding the group operator and partitioning function of Hadoop while our approach employs the most fundamental principle of MapReduce, which is originally supported by Hadoop and has wider adoption. Last but not least, our work also aims at reducing candidate size before computing partial results which are then accumulated at Reduce task. We will show how we make the best use of our filtering strategies in comparison with other work.

3 Preliminaries

3.1 Concepts

Given a set of documents known as worksets $\Omega = \{D_1, D_2, D_3, \ldots, D_n\}$, and each document which has the plain text form contains a set of terms $D_i = \{term_1, term_2, term_3, \ldots, term_k\}$. The symbol [,] represents the list of x elements while the symbol [[,], [,]] indicates the list of lists. In addition, the symbol ‖.‖ denotes the length of a list, i.e., the total number of distinct terms in the set. The most common similarity search problem is to find the document D_j when given a document D_i as a pivot such that D_i and D_j is the most similar pair in Ω. In general, a document D_i can share common terms with other documents in the worksets. In the scope of this paper, we define such common terms as those appearing in all worksets. Furthermore, the inverted document frequency idf_{ik} shows how popular a $term_k$ of a document D_i is compared to other worksets. The inverted document frequency idf_{ik} is computed by dividing the length of worksets ‖ Ω ‖ by the number of documents that also contain the $term_k$. Last but not least, we employ the inverted index as a data structure to quickly support term filtering.

In this paper, we use Jaccard measure, which is widely used and also adopted in [12], to estimate the similarity score of a document pair. Given two sets X and Y, the Jaccard coefficient measuring the similarity between the two sets X and Y is computed as the equation (1) below:

$$sim(X,Y) = \frac{||X \cap Y||}{||X \cup Y||} \quad (1)$$

The value of sim(X, Y) belongs to the interval [0, 1]. If two sets X and Y are similar to each other, the value of sim(X, Y) is close to 1. Otherwise, the value of sim (X, Y) is close to 0.

3.2 MapReduce Paradigm

MapReduce whose details can be seen in [9] is a parallel programming paradigm that helps solve the problem of data scalability. The fundamental idea of MapReduce starting from functional programming languages is focused on two basis jobs named Map and Reduce. The paradigm is deployed on commodity machines in that one plays a role of master and the others take part in as workers. The master node takes its responsibility to assign Map and Reduce jobs to workers. The machines which are assigned Map task are called mappers whereas the machines which are assigned Reduce task are known as reducers. There are M mappers to execute the Map job and R reducers to execute the Reduce job. The Map job is defined by the Map function and the Reduce job is defined by the Reduce function.

A single flow of MapReduce can be briefly summarized as following steps: (1) The input is partitioned in a distributed file system like Hadoop Distributed File System [3] and has the form of [key_1, $value_1$]; (2) Mappers do its assigned tasks from the Map function and produce intermediate key-value pairs of the form [key_2, $value_2$]; (3) The shuffling process from the combine method is conducted to group these pairs according to their key; (4) Reducers receive [key_2, [$value_2$]], apply the Reduce function, and output the final result, which might have the form [result]; and (5) The output is finally written back in the distributed file system.

4 The Proposed Scheme

We start by introducing the simple yet efficient scheme. As illustrated in Fig. 1, the whole process is composed of two MapReduce operations as follows: (1) the first MapReduce operation takes worksets as its input, along with a given query document as a pivot, and then generates a customized inverted index; and (2) the second MapReduce operation exploits the inverted index to compute similarity pairs. Thanks to MapReduce paradigm, each MapReduce phase helps us solve a large volume of data by processing portions of data in parallel. Nevertheless, the more redundant data we eliminate, the better performance we get. On the one hand, the inverted index seems to be a good way to find overlapped sections of each pair. On the other hand, it suffers its long length from useless words that do not contribute much to the similarity score but appear frequently among documents, which makes the inverted index bulky. In addition, such a naïve approach is a waste of computation effort due to generating all candidate pairs and building the full inverted index. Thus, we propose two additional filtering strategies corresponding to each phase, known as Prior Filter and Query Parameter Filter, in order to reduce candidate size leading to eliminating unnecessary pair-similarity computing and avoiding further space cost. In other words, those pairs which are dissimilar to each other should not be computed during the whole process. Doing so partly reduce overheads and improve the performance phase-by-phase.

Moreover, it is noted that intermediate key-value pairs output from REDUCE-1 have the descending order by the length of the list D_i a $term_k$ has and then by the total words W_i of D_i in each list. In this way, we want to apply our filtering methods especially for the pivot document case, with range and k-NN queries in section 5, in order not to transfer much data over the network. As a consequence, the candidate size is significantly decreased.

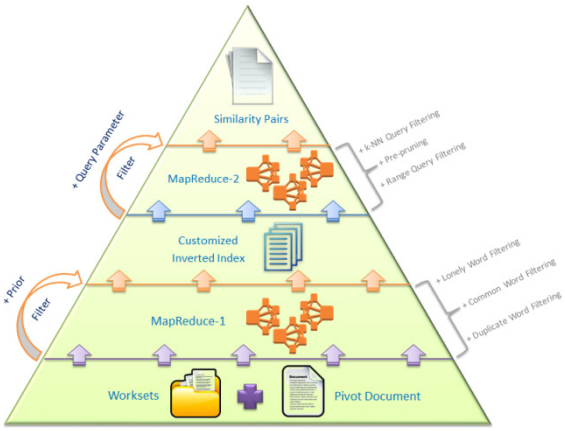


Fig. 1. The overview scheme

The Prior Filter is applied at the MapReduce-1 operation whilst the Query Parameter Filter is attached to the MapReduce-2 operation. The former consists of three sub-filtering methods known as Duplicate Word Filtering, Common Word Filtering, and Lonely Word Filtering. Meanwhile, the latter is composed of another three sub-filtering named Range Query Filtering, Pre-pruning, and k-NN Query Filtering. These filtering methods are alternatively combined to support specific similarity search scenarios. In general, the proposed scheme is not limited to be applied for various similarity search strategies as discussed in section 5 of this paper. For simplicity, we present how the proposed scheme at first works for pairwise document similarity search in sub-section 5.1, and then we show how our scheme is effectively adapt itself to other similarity search parameters in the remaining sub-sections.

Let D_i be the i^{th} document of the workset, W_i be the total words of D_i, n be the accumulated number of the same key, and sim(D_i, D_j) be the similarity score between a document pair. The two MapReduce operations can be summarized as follows:

MAP-1: $[workset] \rightarrow [term_k,\ D_i@W_i]$

REDUCE-1: $[term_k,\ D_i@W_i] \rightarrow [term_k,\ [D_i@W_i]_{ordered}]$

MAP-2: $[term_k,\ [D_i@W_i]] \rightarrow [D_{ij}@W_i@W_j,\ n]$

REDUCE-2: $[D_{ij}@W_i@W_j,\ n] \rightarrow [D_{ij},\ sim(D_i,\ D_j)]$

5 Contributions to Similarity Search

Apart from the general similarity search that does not require any additional arguments, other cases usually come with their own parameters for specific application domains. In this paper, we investigate the most popular ones known as pairwise similarity, the pivot document, k-NN query, and range query. For these cases, we show how adaptable our proposed scheme is by utilizing these provided parameters.

5.1 Pairwise Similarity Case

Pairwise similarity search is also known as self-join similarity search. Documents of the plain text form are considered as the input of the scheme so-called worksets. The mappers from MAP-1 method process the worksets and emit intermediate key-value pairs which have the form of [$term_k$, $D_i@W_i$]. Then, these intermediate key-value pairs are transferred to the reducers from REDUCE-1 method in order to produce key-value pairs of the output form [$term_k$, [$D_i@W_i$]], which is also known as an inverted index. Before producing the output, the Prior Filter is applied to discard duplicate terms, those are common terms having its inverted document frequency value as 0, and those cannot contribute to the pair similarity measures. Thus, discarding these common, duplicate, and lonely words partially help reduce the volume of processing data. As background computing, the Duplicate Word Filtering works with the form [$term_k$, $D_i@W_i$] at each mapper while the Common Word Filtering and Lonely Word Filtering work with the form [$term_k$, [$D_i@W_i@idf_{ik}$]] at each reducer.

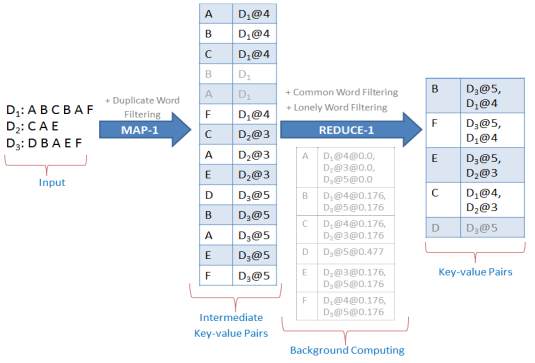


Fig. 2. MapReduce-1 operation

For Example. Assuming that there are three documents named D_1, D_2, and D_3. Each document contains its corresponding words as the input illustrated in Fig. 2. The mappers from MAP-1 method take the input to emit intermediate key-value pairs. Then, they are moved to the reducers from REDUCE-1 method to compute the inverted document frequency for each term. In this example, duplicate terms B and A in D_1, term A whose inverted document frequency is equal to 0.0, and term D which is not shared with the other documents should be discarded. The other terms as B, C, E, and F, whose inverted document frequencies are greater than 0.0, will be emitted as the key-value pairs of the inverted index. In the end of this MapReduce phase, we

finally have the inverted index of the list as following [[B, [D_3@5, D_1@4]], [F, [D_3@5, D_1@4]], [E, [D_3@5, D_2@3]], [C, [D_1@4, D_2@3]]].

After the inverted index is derived from the first MapReduce operation, the second one will be conducted to compute the similarity score. The inverted index goes through the mappers from MAP-2 method which will emit intermediate key-value pairs of the form [D_i, [D_{ij}@W_i@W_j, n]] after shuffling. It is worth noting that the COMBINE function is additionally used to pre-group the intermediate key-value pairs so that the amount of data transferred to the reducers is further reduced. Then the intermediate key-value pairs are retrieved by the reducers from REDUCE-2 method and the final key-value pairs with their similarity scores of the form [D_iD_j, sim(D_i, D_j)] will be computed and ranked according to the scores. Finally, the result can be filtered against query parameters like the threshold similarity ε or the top k-similarity pairs to meet users' search requirements.

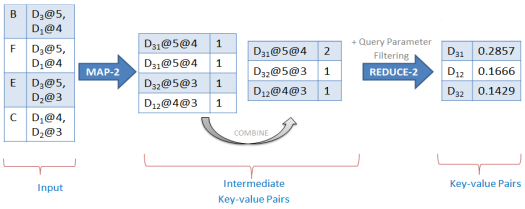


Fig. 3. MapReduce-2 operation

For Example. Fig. 3 indicates the next phase of the previous example. The output of the first MapReduce operation will be considered as the input for the second one. In other words, the inverted index [[B, [D_3@5, D_1@4]], [F, [D_3@5, D_1@4]], [E, [D_3@5, D_2@3]], [C, [D_1@4, D_2@3]]] is processed by the mappers from MAP-2 method to produce the independent intermediate key-value pairs [D_{31}@5@4, 1], [D_{31}@5@4, 1], [D_{32}@5@3, 1], and [D_{12}@4@3, 1]. The combine function groups these pairs according to the key value, and we have the ordered shorter list [[D_{31}@5@4, 2], [D_{32}@5@3, 1], [D_{12}@4@3, 1]]. Then, the reducers from REDUCE-2 method will calculate the similarity score between the pairs. Finally, we have the ordered result list [[D_{31}, 0.2857], [D_{12}, 0.1666], [D_{32}, 0.1429]] which shows the most similarity pair is D_1 and D_3 with the score 0.2857, the pair D_1 and D_2 is with the score 0.1666, and the pair D_2 and D_3 is with the score 0.1429.

5.2 Pivot Document Case

Normally, the query document is issued in order to search other most similarity ones from the database. For instance, given a sample document, we would like to search which documents are the most similar with the provided document. This scenario is so common to most of search applications. The query document is then called the pivot document because we use it as the anchor to compare with others. Once the pivot document is known, we are able to employ it to improve the performance. In this case, the whole follows the general scheme. Nevertheless, we flexibly use Prior Filter between MAP and REDUCE tasks and additionally attach lonely term removal

and pre-pruning processes to the pivot document case. The former takes place at the first MapReduce operation while the latter takes place at the second one.

Those terms which do not appear in the pivot document or those which are contained only in the pivot document will be assessed as lonely terms, and they should be ignored. It is possible because lonely terms do not contribute much to the final similarity score. Therefore, we can approximate the final similarity score by discarding them. When it happens, the inverted index includes terms that appear in the pivot document. On the one hand, we reduce the number of unnecessary terms at the very beginning to reduce overheads. On the other hand, the Jaccard similarity score between the comparing document D_i and the pivot document D_{query} can be re-innovated as the equation below:

$$sim(D_i, D_{query}) = \frac{||D_i||}{||D_{query}||} \quad (2)$$

In the equation (2), $||D_{query}||$ denotes the original length of the pivot document and $||D_i||$ represents the number of $term_k$ it contains. In other words, $\sum term_k$ yields $||D_i||$. The new form of similarity measure is approximately computed by dividing the length of the comparing document by the length of the pivot one. In addition, this new form is utilized by Pre-pruning at MAP-2 to soon eliminate unqualified candidate pairs. More details of Pre-pruning are discussed in section 5.3 for range query case and section 5.4 for k-NN query case. Consequently, those pairs compared with the pivot document and satisfying Pre-pruning constraints are emitted by MAP-2 method.

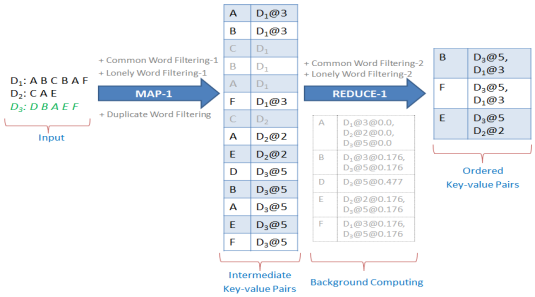


Fig. 4. MapReduce-1 operation with the pivot document D_3

For Example. Considering another example and assuming that D_3 is the pivot document as illustrated in Fig. 4. That means we want to search other most similarity documents with D_3. The whole process is still the same as in the pairwise similarity case except for added lonely term removal, pre-pruning, and the new form of Jaccard similarity measure. In this example, we have $||D_3||$ is 5. At the mapper side, duplicate terms B and A in D_1 are disposed by Duplicate Word Filtering while term C, in both D_1 and D_2, is discarded by Lonely Word Filtering-1 because it is a lonely word that does not appear in the pivot document D_3. Alternatively, Common Word Filtering-1 works with a pre-defined dictionary to pre-filter common words in advance. At the reducer side, term A is removed by Common Word Filtering-2 because it is a common word whereas term D is ignored by Lonely Term Filtering-2 because it is a lonely

word that appears only in the pivot document D_3. After the first MapReduce operation, we have some terms in the list of ordered key-value pairs [[B, [D_3@5, D_1@3]], [F, [D_3@5, D_1@3]], [E, [D_3@5, D_2@2]]]. At the second MapReduce operation as illustrated in Fig. 5, we have D_1, D_2, and D_3 as in turn the candidate similarity pairs. The new form similarity computation shows that the similarity scores are as follows [[D_{13}, 0.4], [D_{23}, 0.2]]. Last but not least, Query Parameter Filtering is optionally used to filter against range query and k-NN query before generating the final output.

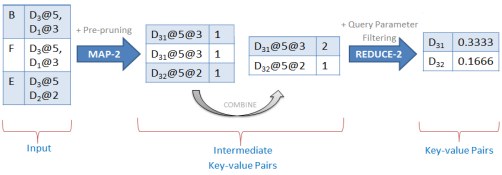


Fig. 5. MapReduce-2 operation with the pivot document D_3

5.3 Range Query Case

From the point of view of range query, a threshold ε is provided in order to find those pairs that have their similarity score greater or equal to the threshold. In general, the result after two MapReduce operations is filtered against ε to find the best fit. In the case of the pivot document, however, we propose to filter unnecessary pairs before their similarity computing. In other words, the final result has to satisfy the equation below which still guarantees the candidate pairs are the super set of the final result:

$$sim(D_i, D_{query}) = \frac{||D_i||}{\left|\left|D_{query}\right|\right|} \geq \varepsilon \tag{3}$$

Because $||D_{query}||$ is computed from MAP-1 method, we call the new upper bound ε' the product between the query threshold ε and the length of the pivot document. As a result, the equation (3) has been equivalently transformed into the equation (4) below:

$$\frac{||D_i||}{\left|\left|D_{query}\right|\right|_{org}} \geq \varepsilon \Leftrightarrow ||D_i|| \geq \varepsilon' \tag{4}$$

$$\text{With } \varepsilon' = \left|\left|D_{query}\right|\right| * \varepsilon$$

It is worth noting that intermediate key-value pairs from REDUCE-1 have the descending order by the length of the list D_i a $term_k$ has. Therefore, we can make the best use of Pre-pruning at MAP-2 method to emit those pairs whose comparing documents have the key value $||D_i||$ greater or equal to the new threshold ε', which partly helps eliminate the amount of unnecessary similarity computations.

5.4 k-NN Query Case

k-NN query looks for the k most similar pairs from the candidate set and can be seen as a special case of range query. Having the same strategy like in the range query

case, we utilize the MAP-2 method to filter candidate pairs against the k parameter before their similarity score is computed. In this case, the mappers from MAP-2 method with Pre-pruning only emit top-k intermediate key-value pairs. For approximate k-NN query, however, each mapper can emit top-pairs whose size is according to the total number of running mappers as the equation (5) below:

$$\underset{for\ each\ mapper}{\text{top} - \text{pairs}} = \max_{k \in N}\left(\left\lceil \frac{k}{\sum Mappers} \right\rceil, 1\right) \tag{5}$$

6 Experiments

6.1 Environment Settings

The experiments are set up in the cluster of commodity machines called Alex, which has 48 nodes and 8 CPU cores and either 96 or 48 GB RAM for each node [2]. The stable Hadoop has the version1.2.1, and DBLP dataset [7] is used to do similarity search on the title of publications. In general, we leave other Hadoop configurations as its default. We want to preserve the most common settings which commodity machines may initially get even though some parameters could be tuned or optimized to fit the Alex cluster. The configured capacity is 5GB per node, so there totally is 240GB for the 48-node cluster. The number of reducers is set to 168. In addition, the replication factor is set to 47. The possible heap size of the cluster is about 629 MB, and each HDFS file has 64MB Block Size. It is worth noting that Alex has suffered the overhead of other coordinating parallel tasks, i.e., these nodes are not exclusively for the experiments. In addition, they are diskless nodes, but in the background data are located on a storage area network. Each benchmark, therefore, is run 10 times to obtain the average values. Last but not least, each benchmark has its fresh running. In other words, data from the old benchmark are removed before the new benchmark starts. All the experiments for one type of query are consecutively run so that their environments are close as much as possible.

6.2 Empirical Evaluation

Fig. 6a represents the query processing time of pairwise similarity case between the naïve approach, which uses MapReduce to compute all possible pairs, and the proposed approach. The difference between two approaches is not so big when the benchmark size is under a specific threshold. The reason is that they have to suffer from the operation cost of the whole system. When the dataset size significantly increases, the query processing time of the proposed method is not so much as that of the naïve method. In other words, when the dataset size is large enough, there is a very big gap between them, which indicates the query processing time of the naïve approach consumes much more while the proposed method outperforms the naïve method in its performance from 5% to 39% when the dataset size grows from 50MB to 500MB. On the other hand, the percentage of saved data volume during the computation phases ranges from 67% to 82%, respectively in Fig. 6b.

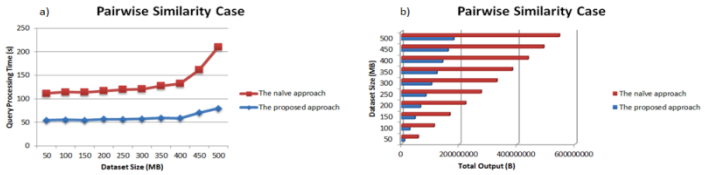


Fig. 6. Pairwise similarity case between the naïve approach and the proposed approach; (a) Query processing time; and (b) The saved data volume

From the point of view of the pivot document case showed in Fig. 7, the dataset size grows from 900MB to 4500MB and mostly reaches the maximum capacity for each node. In Fig. 7a, there is not much difference in performance among all-pairs, range query whose threshold is set to 90% of similarity, and 100-NN query which are based on the proposed methods. Besides, Fig. 7b points out the percentage of saved data volume which will be increased further if query parameters are more given.

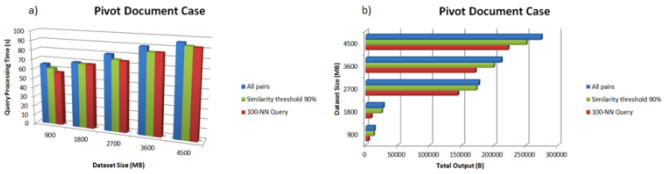


Fig. 7. Pivot document, range query, and k-NN query cases; (a) Query processing time; and (b) The saved data volume

Moreover, what are inherited from the experiments are MapReduce operations should not be too complex due to the lack of memory and the reduction of candidate size is essential because it helps reduce the number of volumes written in HDFS file systems during MapReduce operations, which can lead to achieve high performance.

7 Conclusion and Future Work

In this paper, we propose an elastic approximate similarity search with MapReduce to primarily deal with scalability. We show how our search scheme is specifically tailored for the four most popular similarity search scenarios known as pairwise documents similarity, pivot document search, range query, and k-NN query. In addition, our strategic filtering methods which promote potential scalability of MapReduce help reduce the size of candidate pairs and eliminate unnecessary computations as well as space overheads. Moreover, we conduct experiments with real massive datasets and Hadoop framework to verify these methods.

For our future work, we model worksets as distinct n-grams instead of terms and extend our methods to other metrics to achieve more efficiency. Besides, we also generalize our approach more concretely to the incremental case whose data are on the fly. Furthermore, we concentrate on resolving other factors under the big data context such as the velocity and variety of big data in order to consolidate our methods and look forward to a unified solution supporting data-intensive applications.

References

1. Alabduljalil, M.A., Tang, X., Yang, T.: Optimizing Parallel Algorithms for All Pairs Similarity Search. In: Proceedings of the 6th ACM International Conference on Web Search and Data Mining, USA, pp. 203–212 (2013)
2. Alex Cluster: http://www.jku.at/content/e213/e174/e167/e186534 (referenced on February 4, 2014)
3. Apache Software Foundation. Hadoop: A Framework for Running Applications on Large Clusters Built of Commodity Hardware (2006)
4. Baraglia, R., De Francisci Morales, G., Lucchese, C.: Document Similarity Self-Join with MapReduce. In: Proceedings of the 10th IEEE International Conference on Data Mining, pp. 731–736 (2010)
5. Dang, T.K.: Solving Approximate Similarity Queries. Journal of Computer Systems Science and Engineering 22(1-2), 71–89 (2007)
6. Dang, T.K., Küng, J.: The SH-tree: A Super Hybrid Index Structure for Multidimensional Data. In: Mayr, H.C., Lazanský, J., Quirchmayr, G., Vogel, P. (eds.) DEXA 2001. LNCS, vol. 2113, pp. 340–349. Springer, Heidelberg (2001)
7. DBLP data set, available on, http://dblp.uni-trier.de/xml/ (referenced on March 8 , 2014)
8. De Francisci Morales, G., Lucchese, C., Baraglia, R.: Scaling Out All Pairs Similarity Search with MapReduce. In: Proceedings of the 8th Workshop on Large-Scale Distributed Systems for Information Retrieval, pp. 25–30 (2010)
9. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: Proceedings of the 6th Symposium on Opearting Systems Design and Implementation, pp. 137–150. USENIX Association (2004)
10. Elsayed, T., Lin, J., Oard, D.W.: Pairwise Document Similarity in Large Collections with MapReduce. In: Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies, Companion Volume, Columbus, Ohio, pp. 265–268 (2008)
11. Fenz, D., Lange, D., Rheinländer, A., Naumann, F., Leser, U.: Efficient Similarity Search in Very Large String Sets. In: Ailamaki, A., Bowers, S. (eds.) SSDBM 2012. LNCS, vol. 7338, pp. 262–279. Springer, Heidelberg (2012)
12. Li, R., Ju, L., Peng, Z., Yu, Z., Wang, C.: Batch Text Similarity Search with MapReduce. In: Du, X., Fan, W., Wang, J., Peng, Z., Sharaf, M.A. (eds.) APWeb 2011. LNCS, vol. 6612, pp. 412–423. Springer, Heidelberg (2011)
13. Szmit, R.: Locality Sensitive Hashing for Similarity Search Using MapReduce on Large Scale Data. In: Kłopotek, M.A., Koronacki, J., Marciniak, M., Mykowiecka, A., Wierzchoń, S.T. (eds.) IIS 2013. LNCS, vol. 7912, pp. 171–178. Springer, Heidelberg (2013)
14. Vernica, R., Carey, M.J., Li, C.: Efficient Parallel Set-similarity Joins Using MapReduce. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, USA, pp. 495–506 (2010)
15. Xiao, C., Wang, W., Lin, X., Yu, J.X.: Efficient Similarity Joins for Near Duplicate Detection. In: Proceedings of the 17th Int'l World Wide Web Conference, pp. 131–140 (2008)
16. Zhang, D., Yang, G., Hu, Y., Jin, Z., Cai, D., He, X.: A Unified Approximate Nearest Neighbor Search Scheme by Combining Data Structure and Hashing. In Proceedings of the 23rd International Joint Conference on Artificial Intelligence, pp. 681-687 (2013)

Standardized Multi-protocol Data Management for Grid and Cloud GridRPC Frameworks

Yves Caniou[1], Hadrien Croubois[2], and Gaël Le Mahec[3]

[1] Université de Lyon, JFLI CNRS, Japan
Yves.Caniou@ens-lyon.fr
[2] Université de Lyon, ÉNS Lyon
Hadrien.Croubois@ens-lyon.fr
[3] Université de Picardie Jules Verne, MIS Laboratory, France
Gael.Le.Mahec@u-picardie.fr

Abstract. GridRPC is an international standard of the Open Grid Forum defining an API designed to allow applications to be submitted in a seamless way on large scale, heterogeneous and geographically distributed computing platforms. First versions of the standard did not take into account any data management feature. Data were parameters of the Remote Procedure calls, without any possibility to prefetch them, to use persistence, replication, external sources, *etc.* , and making GridRPC codes middleware dependent. The data extension of the standard introduced a short set of functions and data structures to complete the API with simple but powerful data management features. In this paper, we present a modular and extensible implementation of both APIs, which needs only a few developments to be usable with any middleware relying on RPC, and which provides access to numerous and easy to extend protocols and data middleware to access data. Gaining data management functions, it introduces interesting potentiality for optimization that such an approach would provide to large scale applications.

1 Introduction

Many applications use RPC-like mechanisms to distribute computations over nodes of clusters and supercomputers composing some distributed systems like a grid, a cloud, or both (now referred as sky computing). Combined with connections to huge databases, they more or less transparently provide scientists with the possibility to focus on their core thematic, giving them more time to deal with data analysis, without dealing with the underlying complexity of all the different mechanisms involved into job and data management. More lately applications even directly couple analysis, graphical representations and such, making platform management only a part of their project, whose actions are generally available through some web site. And surprisingly, when considering a new area, a new platform, new independent pieces of software are often developed instead of using previous work, software or standardized APIs.

The Open Grid Forum standard defining the GridRPC paradigm, namely Remote Procedure Call over the Grid, has been published in 2007, benefiting

A. Hameurlain et al. (Eds.): Globe 2014, LNCS 8648, pp. 61–72, 2014.

from 10 years of experience by their respective authors. Simple and easy to use, it has been completed with a standardized data extension only recently. This extension to the native API proposes to expert users to easily handle remote data and to optimize distributed applications with prefetch, migration or replication of possibly distant data using multiple asynchronous transfers together with remote procedure calls on available distributed computing resources.

Based on preliminary experiments[1,5], applications also benefit from multi-administration sites resources managed by multi-middleware (inherent to interoperability provided with the implementation of the API data extension) and target not only traditional Grids but any distributed platform possibly composed of resources from the Cloud [6].

In an attempt to simplify and develop interoperability, and to unify previous works, we propose here a library managing *both* GridRPC and GridRPC Data Management APIs. We present an overview of the project architecture, designed with a very modular prospect, relying on middleware and data manager modules but also bringing inner data manager capabilities and transfer protocols. Having in mind not to go too much into details, we highlight here some of its features, such as the asynchronous requests management and the transfer management, which involves mapping and scheduling aspects: there is interesting potentiality for optimization at the data operation level, with scheduling to reduce the completion time of a data operation when several sources and several destinations are provided but not necessarily interconnected; and at the workflow/dataflow level to reduce any [sub part of an] application graph. At the moment, the library provides modules for the grid middleware DIET and Ninf , and data manager modules for projects and protocols like `Dagda`, `iRods`, `webdav` (used for web-based repositories like dropbox, owncloud), `ftp` and `rsync`.

The rest of the paper is organized as follows: next section explains the motivations behind the GridRPC DM API and some related work. Section 3 presents the global design of the implementation, the different issues that the API leads to and their solution. Section 4 presents some validation experiments and after explaining some future work directions, we conclude in Section 6.

2 State of the Art

2.1 The GridRPC Data Management API, Summary

The GridRPC DM API [2] introduces the concept of data handle and with it, several GridRPC data types to provide standardized information, for example lists of input and output URIs to give the locations of respectively source and destination [remote] data, with the according protocols to access it at the considered location). It also defines mode managements for a client to characterize the persistence of the data in the system, *etc.* All actions (initializing, transferring, waiting for completion of asynchronous transfers, *etc.*) are provided with only 12 functions.

This standard answers at the API level to issues related to **feasibility** of the computation by decoupling the data from its locations and from protocols

to access it; to **performance** using different sources and protocols to access a remote data, providing explicit data management with the possibility to prefetch and to migrate data, as well as the possibility to rely on some smart middleware to transparently handle data management; and to **extensibility** by providing containers of data. It also solves **portability**, making GridRPC codes portable from one middleware to another.

2.2 Related Work

Similar works can address some data management issues in the GridRPC but only separately and without integration into remote procedure call: one can store data on a distributed file system like GlusterFS[1] or GFarm [9] to deal with automatic replication; OmniRPC introduced omniStorage [7] as a Data Management layer relying on several Data Managers such as GFarm and Bittorrent. It aims to provide data sharing patterns (worker to worker, broadcast and all-exchange) to optimize communications between a set of resources, but needs knowledge on the topology and middleware deployment to be useful; DIET also introduced its own data managers (DTM and Dagda [3,4]), which focus on both user explicit data management and persistence of data across the resources, with transparent migrations and replications.

At a higher level, Stork [8] is a batch scheduler specialized in data placement and data movement. If the transfer protocol specified in the job description file fails for some reason, Stork can automatically switch to any alternative protocol available between the same source and the destination hosts and complete the transfer; Galaxy[2] is a web interface written in python allowing on-line design of task workflows. Galaxy focuses mainly on bioinformatics but could be used for all type of applications relying on workflow execution. By default Galaxy is configured to execute application on its host server but can use the OGF DRMAA API to distribute computations on remote servers. Data can only be transferred as files. On the contrary of classical RPC, there is no simple way to upload data directly on the application memory address space. Moreover, the GridRPC API modularity allows to combine simplicity of such data management systems and tunability by choosing where and when data are transfered.

By using standardized GridRPC code with our implementation and its corresponding modules, it should be possible to benefit at a upper layer from previous works, gaining in portability and interoperability with middleware and data managers, which in turn provides access to a potentially larger set of resources and architectures.

3 Implementation: Architecture and Features

We present in this section the system underlying our implementation of the GridRPC and GridRPC Data Management standards. We highlight the features

[1] `http://www.gluster.org/`
[2] `http://galaxyproject.org/`

of the library, its data management capabilities as well as scheduling possibilities between and for each data operation, *i.e.,* the set of all transfers requested between the URIs provided as sources and destinations for the same data.

The library is developed in C++ and C, using internally `boost`, and `cmake` to build the project. It is freely available from a sourceforge repository: `http://sourceforge.net/projects/gridrpcdm/`.

3.1 Modularity of the Solution

The proposed implementation can be viewed as a *meta-implementation* of the APIs (see Figure 1) since it provides the two GridRPC APIs, adding some seamless mechanisms for performance (scheduling *etc.*) in a middleware and protocol "agnostic" manner. The library does not interact directly with the middleware nor the data storage servers. It proposes a fully interoperable API for any middleware and protocol/data manager with only very few specific developments of simple modules. The module developers do not have to take care about which data transfer protocol is available, like the data manager module developpers do not have to care about which middleware is used to call remote procedures. To do so, different interfaces are provided by the library:

- The client application interface: the external client API. Clients can use the API directly without any knowledge about the underlying GridRPC middleware. However, by adding a prefix to the service name, users can force the library to use a specific middleware (*e.g.,* `"DIET:matmul"` and `"Ninf:matmul"` select respectively DIET and Ninf-G for the `"matmul"` service).
- The Services interface: it is a subset of the client API with some additional utility functions facilitating servers conversions from standard GridRPC servers to GridRPC Data Management servers.
- The Modules interface: the library defines a set of functions that should be exposed by the module to extend the library capacities.

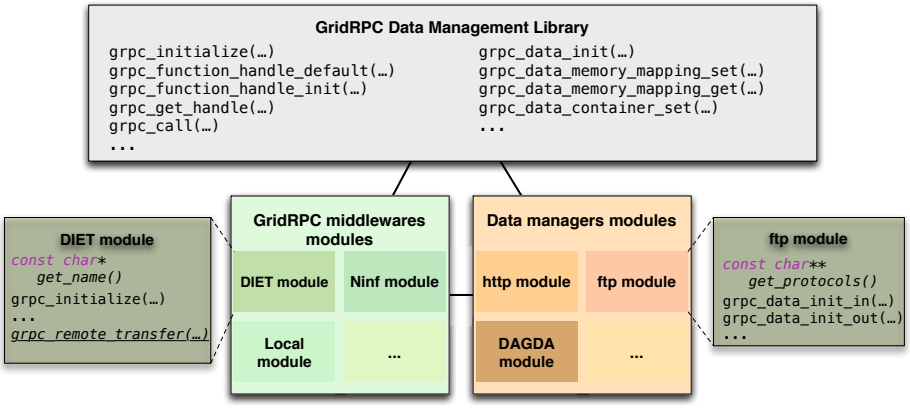


Fig. 1. A very modular implementation

Integrating a new Middleware requires to fulfill a set of 10 main functions and one optional. Most of them are just type conversions functions from the existing middleware data-type to the new GridRPC data-types. The most complex function of a middleware module is `grpc_remote_transfer()` which initiates a transfer from a remote host to another remote host. A default implementation is included in the library relying on a middleware service call: the module developpers have just to implement this simple service using the library transfer capabilities on the server side.

Remarks:

- To avoid "name conflicts" between existing GridRPC implementations and the new definitions of the library, definitions in the library headers files are automatically prefixed when needed, allowing an easy reuse of the existing functions without name-clashing at the compilation step.
- Note on asynchronous calls: they are internally managed by the library from synchronous calls to middleware. However, middleware functions must be reentrant for a safe asynchronous use.

At the moment, the modules for the DIET and Ninf GridRPC middleware are available.

Integrating a New Data Manager Module requires to provide 4 functions: 2 initialization functions corresponding to input and output data, which can most of the time be left empty; and 2 transfer functions to get and put a data. They are generally wrappers of existing transfer protocol libraries (*e.g.,* `libcurl` for `http` and `ftp`).

At the moment, the library implements the data manager modules for `rsync` and `scp`, using the shell commands; `iRODS`, using the shell command (the library is only available for Java and PHP); `webdav`, to access `Owncloud` and `Dropbox` servers. It uses the `neon` library and; `curl`, to access `http` and `ftp` via the `curl` library.

Module Initialization. The library global initialization process reads the global configuration file to determine which module should be dynamically loaded at execution time, where to find it, and some parameters available for each module in its own separate section.

The initialization function of each module is then processed sequentially, passing the arguments of the module specific configuration, and potentially reading more parameters in the deployed module-specific configuration file.

3.2 Asynchronicity Management

We call a request the inner action managed in the library: they correspond to API calls for remote procedure, API calls for a transfer or a group of transfers involving a unique data. For example when one source and several destinations

are provided as input and ouput URIs, several transfers are involved in group to provide the unique API transfer call. All requests are managed the same way by the request controller: this entity registers each of them during their initialization, and with the help of threads and semaphores it limits their number and immediately knows the idendity of a request that completes without active wait. Some additional dependency information is also recorded with each request, and thus a hierarchy of requests (the link being the temporal dependency) can be built. It is used to express the concept of a group of requests reported above in the transfer example, but it is also a powerful way to handle waits for one or a group of asynchronous remote procedure calls as well.

Requests are managed with a priority system, which has been instantiated in the current implementation with a queue managed with a FIFO algorithm and a limitation on the number of parallel threads executed at a given moment: There is not much more that can be done at the moment: since there is no dependency information between data transfers operated at the API level, one cannot try any optimization between requests that do not belong to the same group because it could generate inconsistency in data or failure. However coupled with a system that handles workflow/dataflow, some meta-scheduling over available GridRPC middleware and data managers may be performed.

3.3 Data Manager Capabilities

The library does not only operate with underlying data transfer projects. It must provide the data persistence as defined in the API, integrate the possibility to communicate in-memory data (which possibly avoids at least one copy to disk), and make the junction between different locations where the data is available, and the protocols with which one can access them. The latter induces possible hidden (automatic and mandatory) copies and scheduling for the data to be transferred to all requested destinations.

Data Persistence. The GridRPC Data Management API defines numerous persistence modes: the data can be volatile, *i.e.,* there is no special requirement on its management and this can be considered as the default mode; it can be strictly volatile, meaning that the library has to provide means to remove the data from the platform after a computation (thus some protocols and data managers cannot be used); when defined as sticky, the data or a copy must be kept on the location where the client requests are executed; if unique sticky, no replication nor migration can be performed; finally, the client can also request the library to transparently manage prefetch, replication and migration of data. Then, by also handling procedure calls the library can perform some scheduling in order to reduce some metrics. At the moment persistent data are managed through DAGDA.

The `Memory` Protocol. Each data is referenced by a given set of specific URIs, providing the transfer protocol or the underlying data manager to use (for example `http` or `dagda`). But when trying to get performance, on linear algebra

computation for example, there is a need to keep data in memory and avoid file transfers. The GridRPC Data Management API foresaw this kind of use and introduced the `memory` protocol. In addition to this protocol management, our implementation lets a client (or the library itself) use URIs with query and fragment. This leads to possible evolution for improvements (see after) and to manage more data managers (like P2P middleware that initiate torrents with specific files).

Implementing the `memory` protocol means that the library has to use GridRPC middleware inner data manager which can hopefully communicate between its own components to achieve such a need. But when a data is in memory and has to be transferred either on another GridRPC middleware components or on a storage server, it has to be written to a file and then be manipulated (transferred and possibly handled remotely) to be in the requested status. This part uses (de)serialization functions, defined by the GridRPC Data Management API, partly relying on tools provided by the `boost` library. But that maybe shows an unclear part of the API: the protocol to use in that specific case to manipulate the file is not precised. In our current implementation, the protocol is static and is read from the middleware configuration file at initialization time.

But if going a bit further than the API, we can use the query part of the URI. Indeed considering a data available in memory, the API does not provide a mean to know which protocol(s) can be used to send or to receive it since the URI would be similar to `memory://graal.ens-lyon.fr/matrixA`. In ongoing work, our library is going to explore what can be done with specifying protocols within the URI query part, *e.g.,* `?protocol=rsync?protocol=webdav`.

Scheduling for Implicit and Explicit Data Transfers. Data transfers are operated 1) when data participate to a remote procedure call. They are in that case implicit or automatic, and; 2) can be explicitly requested by a client with a call to `grpc_data_transfer()`.

Implicit and Automatic Transfers: When a remote procedure call is performed, meta-data are serialized and transferred to the distant service. They contain sets of URIs which may lead to additional transfers before and after the service execution (Fig 2).

- If one of the input URI refers to a memory or file data on the client, the data must be available to the service before its execution so that it can remotely access it.
- If one of the ouput URI refers to a memory or file data on the client, the service must have made it available and the client must get it.

Explicit Data Transfers: Several transfers are operated by `grpc_data_transfer()`, *i.e.,* a call to an explicit transfer operation: the data should be present in all locations set in the input URIs list, and must be present in all locations set in the output URIs list. This can be treated with a sequential set of transfers from one given source to each destination for

example, but that would be inefficient. In addition, it is not mandatory that all participants are directly interconnected (either by network or by protocol) and transfers may have to be scheduled to make the whole operations possible (destinations of completed transfers being considered as potential sources). The library also makes possible to delegate transfers on all GridRPC servers that offers some library specific service. Hence transfers can be distributed over nodes to reduce the bandwidth impact, and/or to try to reduce the transfer operation completion date for example.

In order to build a schedule in our implementation (made by the dispatcher, Fig 3), we list the nodes that can participate to a transfer operation: To our benefit, since the library contains middleware modules, it can also rely on underlying GridRPC middleware to potentially add relay servers to [remotely] distribute the transfer load or a part of it. To discover those middleware nodes, the library provides an `echo` service, that must be deployed, *i.e.,* registered in the GridRPC server capabilities (at the moment, only middleware nodes with the `memory` protocol available are considered. If the service is not deployed, the node is simply not considered as a possible relay).

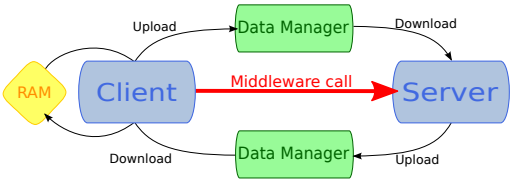


Fig. 2. Automatic transfers during a GridRPC call

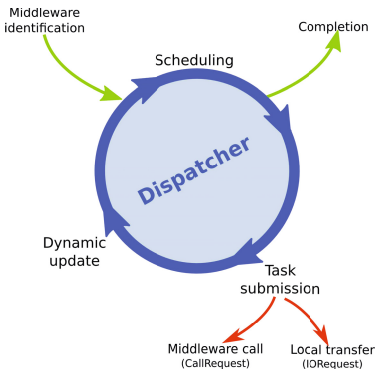


Fig. 3. Dispatcher's cycle

Then URIs are sorted: Local, Middleware node or Storage server; and the dispatcher uses a Round-Robin algorithm to build and launch every one-to-one transfer according to the sorted list below: the list describes *by priority of action*, matching one input URI to one output URI depending on their nature (Local, Middleware or Storage), the action undertaken to manage the corresponding transfer. As seen in Sec. 3.2, transfers at the same time are possibly limited in number, they are monitored with an effective and sufficient semaphore mechanism, and a completion leads to a dynamic update of the set of input URIs. The above algorithm loops until all transfers to ouput locations are done. In case of failure due to an unresponsive input middleware, the middleware is not considered anymore in the next scheduling/mapping cycle.

L-S: The transfer is initiated locally.
M-S: The transfer is processed through a call to the remote transfer service.
S-L: The transfer is initiated locally.
S-M: The transfer is performed through a call to the distant transfer service.
L-L: The transfer is initiated and performed locally.
L-M: The library makes the local data available via the GridRPC middleware inner data manager whose remote counter part will download afterwards.
M-L: The remote middleware is being asked to make the data available, this data is then downloaded by the GridRPC middleware.
M-M: The source middleware is asked to make the data available so that the destination middleware can download it when needed.
S-S: The library first tries to invoke a remote service on the destination server to initiate the transfer. If the transfer fails, data is downloaded on the library client, then transferred to the destination server. If there is no available protocol to proceed to such transfers, the call fails returning an error code.

4 Experimental Results

4.1 Multi-protocol and Dispatcher Scheduling/Mapping Validation

Table 1 lists the experiment deployment. We used 3 computing resources, 2 in Japan and 1 in France, on which we deployed `iRODS` and `ssh` servers, and DIET components: a client, a dietAgent (the registry), and a server (matrix addition) written with the GridRPC APIs requirements together with our library. Two matrices are defined with a list of input URIs depending on the running test, described hereafter.

Table 1. Resources involved in Experiment 1

Machine (location)	Services	Data (protocol)
Arcterix (JFLI - Japon)	dietAgent, client, `sshd`	matA (`ssh`)
yume (JFLI - Japon)	service '`+`', `iRODS`, `sshd`	matA, matB (`ssh`)
graal (Éns-Lyon - France)	`sshd`	matB (`ssh`)

There are four tests, built with the scenario of getting the two matrices through `ssh`, performing the addition, and uploading the result to an `iRODS` server (here locally):

- Remote/Remote: the client does not include the URIs concerning the host *yume* in the input list used for the remote call.
- Remote/Local: the client does not use the URI concerning matA on *yume* in the input list used for the remote call.
- Local/Remote: the client does not use the URI concerning matB on *yume* in the input list used for the remote call.
- Local/Local: all URIs are used for in remote call.

This simple experiment aims to show both 1) the seamless multi-protocol management of the library, as well as 2) the possibility for the dispatcher, described page 68, to perform a schedule: due to its priority matching combined with its Round-Robin algorithm, the library uses the local data first if available. Figure 4 clearly shows this behavior, the blue region showing the time spent during each transfer when it occurs.

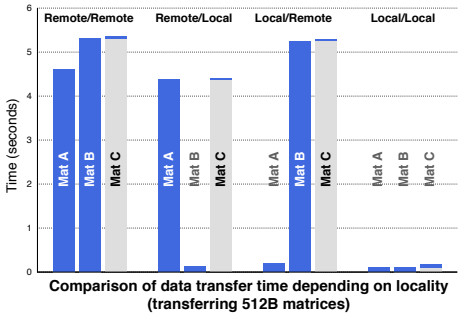


Fig. 4. Results for Experiment 1

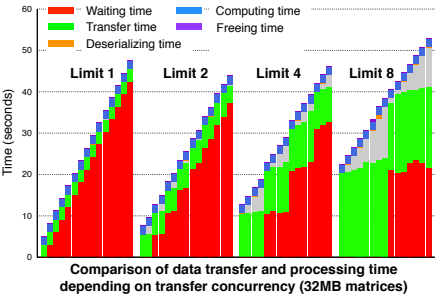


Fig. 5. Results for Experiment 2

4.2 Asynchronous Transfers Management and Bounded Number of Simultaneous Transfers

For this experiment, we designed the following scenario: a remote procedure call is performed to add a given number of matrices which are available remotely through `ssh`. Matrices are downloaded and added as soon as the operation is possible, *i.e.,* at first when two of them are finished to be downloaded, then every time a new one has been downloaded and the previous computation has finished. We performed 4 tests, corresponding to the number of simultaneous transfers that it is possible to make at a given instant, resp. 1 unique transfer, 2, 4 and 8 transfers maximum at a given time. The number of matrices is fixed to 16, and one matrix is 32MB (*i.e.,* 2000×2000 of 64bits integers).

We designed this experiment to validate the request controller behavior, *i.e.,* the implementation of the possibility to limit the number of simultaneous transfers occurring in a transfer operation, and the possibility to use the waiting functions, here with `grpc_data_transfer()` and the `GRPC_WAIT_ANY` parameter, making the server able to perform an operation as soon as enough matrices are present on the server side (the addition was chosen for the operation since it requires less time than a transfer, which leads to show the wanted behavior. Besides, it also makes sense conceptually since it's a commutative operation).

Figure 5 shows the activity of the service and its duration on the y-axis, related to the progression over the number of matrices downloaded for each limit on the x-axis. The same evolution by group of cardinal equal to the possible limit of both the waiting time (non-active wait) and transfer time highlights that the number of simultaneous transfers operated by the library is indeed configurable

(for the moment the information is static in the configuration file. We intend to look if it makes sens to have it self-tuned by the library, depending on the dynamicity of both the network and computing performance). It also shows that every computations occur when enough matrices are finished to be downloaded. As a side effect, it also confirms the observations made in [8]: there's a real need to limit the number of possible parallel transfers. We can indeed observe on this small example that the overall completion time of the addition of the 16 matrices is a bit reduced when the limit is fixed to 2 for our small testbed.

5 Future Works

Future works are heading towards different directions. If the library is already usable and implements most of the API, more performance can be obtained with more efficient scheduling: at the request controller (Section 3.2), and at the dispatcher level (Section 3.3); and more development: for example including a middleware module for `ssh` would add more scheduling possibilities; the protocol `memory` leads to already complex data management mechanisms, yet to be continued in addition to a `file` protocol that would help avoiding useless data copies, making the use of the library even more scalable. Modules for dCache and GridFTP would possibly make transfers faster, but further control would have to be done on the bandwidth consumption; a data manager module for Amazon S3[3] would give further access to cloud storage resources leading for a need to also take into account some financial criteria in the above scheduling process, and possible migration of data when possible (*e.g.,* when the data is requested as GRPC_PERSISTENT).

6 Conclusion

With the GridRPC Data Management standard completing previous works on GridRPC, both at the API and software level, feasibility of computations and performance is at reach with immediate portability and interoperability between GridRPC middleware. To ease its spread, while giving access to GridRPC middleware and to existing data managers, we provide an implementation of both APIs relying on a very modular architecture. Fulfilling the standard requirements, the library also implements the data management modes as well as a `memory` protocol to avoid useless copy to disk. We showed that an efficient system to handle waiting mechanisms is in place and that we operate some mapping/scheduling when several transfers are involved in the same data management. We conducted some experiments and obtained results validating the expected behaviors. From now on we will focus on more theoretical work to improve the yet non-trivial mapping/scheduling of transfers involved for a given data, and we are considering to plug a workflow/dataflow analyzing tool to schedule transfers of different data with remote procedure calls altogether.

[3] `http://aws.amazon.com/`

Further developments will also occur, giving more adaptability and choices to the end-user while bringing new issues concerning scheduling possibilities, for example with Cloud storage resources.

Acknowledgment. This work is partially founded by the ÉNS Lyon. The authors want to thank Hidemoto Nakada for the Ninf middleware module.

References

1. Caniou, Y., Caron, E., Mahec, G.L., Nakada, H.: Transparent Collaboration of GridRPC Middleware using the OGF Standardized GridRPC Data Management API. In: The International Symposium on Grids and Clouds (ISGC), February 26-March 2. Proceedings of Science, 12 p. (2012)
2. Caniou, Y., Caron, E., Mahec, G.L., Nakada, H.: Data management API within the GridRPC. In: GFD-R-P, vol. 186 (June 2011)
3. Del-Fabbro, B., Laiymani, D., Nicod, J.M., Philippe, L.: DTM: a service for managing data persistency and data replication in network-enabled server environments. Concurrency and Computation: Practice and Experience 19(16), 2125–2140 (2007)
4. Desprez, F., Caron, E., Le Mahec, G.: DAGDA: Data Arrangement for the Grid and Distributed Applications. In: International Workshop on Advances in High-Performance E-Science Middleware and Applications, AHEMA 2008, Indianapolis, Indiana, USA. In conjunction with eScience 2008, pp. 680–687 (2008)
5. Camillo, F., Caniou, Y., Depardon, B., Guivarch, R., Mahec, G.L.: Improvement of the data management in GridTLSE, a sparse linear algebra expert system. JCIT: Journal of Convergence Information Technology 8(6), 562–571 (2013)
6. Muresan, A.: Scheduling and deployment of large-scale applications on Cloud platforms. These, Ecole normale supérieure de lyon - ENS LYON (December 2012)
7. Nakajima, Y., Aida, Y., Sato, M., Tatebe, O.: Performance evaluation of data management layer by data sharing patterns for GridRPC applications. In: Luque, E., Margalef, T., Benítez, D. (eds.) Euro-Par 2008. LNCS, vol. 5168, pp. 554–564. Springer, Heidelberg (2008)
8. McLaren, J., Kosar, T., Hutanu, A., Thain, D.: Coordination of access to large-scale datasets in distributed environments. In: Shoshani, A., Rotem, D. (eds.) Scientific Data Management: Challenges, Existing Technology, and Deployment. CRC Press/-Taylor Francis Books (2009)
9. Tatebe, O., Hiraga, K., Soda, N.: Gfarm grid file system. New Generation Computing 28, 257–275 (2010)

Improved Recovery Management and Routing in W-Grid, a Distributed Infrastructure for Effective and Efficient Multidimensional Data Management over Wireless Ad-Hoc Sensor Networks

Alfredo Cuzzocrea[1], Gianluca Moro[2], and Claudio Sartori[3]

[1] ICAR-CNR and University of Calabria, Italy
cuzzocrea@si.deis.unical.it
[2] DISI Department – Cesena Branch, University of Bologna, Italy
[3] DISI Department – Bologna Branch, University of Bologna, Italy
{gianluca.moro,claudio.sartori}@unibo.it

Abstract. In this paper we focus on data management aspects of W-Grid, a decentralized infrastructure that self-organizes wireless devices in an ad-hoc manner where each node has one or more virtual coordinates through which both message routing and data management can be combined and performed in a cross-layer fashion. Differently from existing solutions, W-Grid does not require complex devices that need global information or external help from systems, such as the *Global Positioning System* (GPS), which works only outdoor with a precision and an efficacy both limited by weather conditions and obstacles. Therefore our solution can be applied to a wider number of scenarios, including mesh networks and wireless community networks. In particular, in this paper we introduce two extensions to W-Grid: (*i*) recovery capabilities to network or node failures without using broadcasting operations and (*ii*) improved routing based on a local learning with a new method of evaluating logical distances among nodes through implicit cost-free overhearing at sensors.

1 Introduction

A wide number of routing algorithms for ad-hoc wireless sensor networks have been proposed in the literature, ranging from those that adopt message broadcast/flooding to those using *Global Positioning System* (GPS) to discover the routing path towards the destination. Broadcast algorithms, while simple to implement, are not scalable due to the enormous overhead caused by congestion in large networks. On the other hand, solutions based on GPS, which rely on exact geographic position for each node, do not work in indoor environments and do not function correctly in extremely dense networks or in adverse climatic conditions. Technical and economic feasibility constraints also prevent from attaching a GPS receiver to each node in very large network (i.e. made of thousands

A. Hameurlain et al. (Eds.): Globe 2014, LNCS 8648, pp. 73–84, 2014.

of devices). For these reasons our solution does not rely on GPS or any other positioning system. The routing problem has also been addressed in cases of both total absence and partial availability of geographic location information by generating virtual coordinates to approximate real ones. Our solution may be classified within this set of approaches in that it also uses virtual coordinates, but it is different in that it does not aim to approximate real coordinates, but rather it *simulates* them.

W-Grid [1,2] is a distributed binary tree index cross-layering both routing and data management features, in that (1) it allows efficient message routing and, at the same time, (2) the virtual coordinates determine a data indexing space partition for the management of multi-dimensional data. Each node has one or more virtual coordinates on which an order relation is defined and through which the routing by content occurs; each virtual coordinate represents a portion of the data indexing space for which a device is assigned the management responsibility. A consequence of this approach is that nodes which are close in the physical network topology are also close in the logical overlay network.

In this paper we focus on data management aspects of W-Grid, and we introduce (i) a new recovery method that does not require broadcast messages and (ii) a new routing approach based on a local learning method that improves also the traffic balancing among nodes without affecting energy consumptions. The solution does not require GPS because each device receives a virtual coordinate reflecting its local connectivity with other neighbor devices and each of them uses this information to perform routings and to search for data; the data management is performed natively, namely in a cross-layer fashion, thanks to the fact that each device receives a set of unique virtual coordinates, each of which represents also a portion of the data indexing space for which a device is assigned the management responsibility. We also show that W-Grid is at some extent robust to sensor failures, meaning that if a sensor or a link crashes or turns off, neighbor sensors are able to recover the network failure without using local broadcasting, when physically possible, i.e. when the failure does not cause the partition into unconnected subnetworks. In this work we consider W-Grid to be used in wireless ad-hoc and sensor networks, therefore nodes disconnections are basically represented by failures (e.g. power exhaustion).

With respect to previous work, in [17] we described a preliminary W-Grid version with routing and data management features, in [22] we introduced data replication to allow faster content location while in [2] we presented the range query capability of W-Grid. In [1] we introduced a lazy recovery algorithm to resolve in background possible routing problems while in [2] the infrastructure has been extended with reactive recovery capabilities to solve node failures as soon as they are detected; however, differently from the solution presented in this paper, both recovery solutions required some broadcast operations. Finally a preliminary version of routing with learning capability is reported in [23].

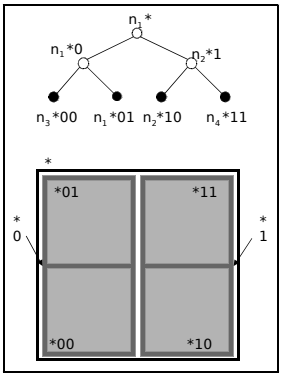


Fig. 1. Correspondence Among Coordinates and Multi-Dimensional Data Space Partitions

2 W-Grid Overview

From now on we will refer at devices with the terms sensors or nodes indistinctly. Basically W-Grid can be viewed as a binary tree index cross-layering both routing and data management features in that, (1) implicitly generating coordinates and relations among nodes allows efficient message routing and, at the same time, (2) the coordinates determine a data indexing space partition for the management of multi-dimensional data. Each node has one or more virtual coordinates on which an order relation is defined and through which the routing occurs, and at the same time each virtual coordinate represents a portion of the data indexing space for which a device is assigned the management responsibility. Differently from algorithms based on geographic routing, W-Grid routing is not affected by dead-ends. Since in sensor networks the most important operations are data gathering and querying it is necessary to guarantee the best efficiency during these tasks.

While W-Grid architecture and main functionalities have extensively be presented in [1,2], in this paper we further focus on data management aspects of W-Grid.

W-Grid distributes data (represented as tuples of attributes) gathered by sensors or shared by nodes among them in a data-centric manner. Values are linearized into binary strings (see [24]) and are each stored at the nodes/sensors whose W-Grid coordinates have the longest common prefix with the resulting strings. Thus, a W-Grid network acts directly as a distributed database and coordinates c are used as a roadmap to data repository. This means that each coordinate represent a portion (i.e. region) of the global data space as depicted in Figure 1. Regions are generated according to data distribution and the use of a bucket size for each data region, together with a load balancing algorithm, allow to balance nodes storage load [17].

Let us describe a brief example of an environment monitoring application in which sensors survey temperature (T) and pressure (P), to which we refer as d_1

and d_2. Each event is inserted in the distributed database implicitly generated by W-Grid, reporting for instance date and time of occurrence. Without loss of generality we can define a domain for T and P let us say $Dom(d_1) = [-40, 60]$ and $Dom(d_2) = [700, 1100]$. We present two examples: (i) an exact-match and (ii) a range query submitted to the network.

(i) Return the times at which sensors surveyed a temperature of 26 Celsius degrees and a pressure of 1013mbar. The linearization [25,24] of the two-dimensional data values results in a binary string which indicates the path to be followed in the network to get to the sensor storing the data. Then, any sensor can be taken as starting point for the query to get to the destination. In this case the result of the linearization is[1]:

$$c_t = *11011000$$

As described in [25,24] the length of the destination string can be adjusted, without affecting the hops that were previously covered, during the routing if we find that sensors with longest string exist.

(ii) Return the times at which sensors surveyed a temperature ranging from 26 to 30 Celsius degrees and pressure ranging from 1013 to 1025mbar. After calculating the correspondent binary string for the four corners of the range query, namely:

$$\begin{array}{c} (26,1013)\ (26,1025)\ (30,1013)\ (30,1025) \\ c_1 = *11011000\ c_2 = *11011001 \\ c_3 = *11011010\ c_4 = *11011011 \end{array}$$

all we have to do is querying sensors whose coordinates have $*110110$ as prefix.

3 Active Recovery of Node Failures

In ad-hoc networks nodes usually have scarce resource and they especially suffer of power constraints. This can lead to nodes failures that could affect routing efficiency. In W-Grid robustness is guaranteed by multiple coordinates at each node and by the adopted routing metric. In fact, it is possible to route through different paths. If a broken path is discovered the packet can change direction (e.g. next hop) and follow a different path, according to another coordinate. Whenever a nodes detects that it can not contact one of its father(s) it must start a recovery procedure and find its closest existing relative. In [17] we forced the orphan node to perform a "local broadcast" searching for the parent of the missing coordinate. The term "local broadcast" was used since it is very likely that the searched coordinate will be close to the broadcasting node considering that it is a close relative. However, even if the broadcast packet time-to-live is

[1] By standardizing 26 and 1013 to their domains we get 0,76 and 0,78 respectively. We multiply both of them by 2^4 to get a string of length 8. The binary conversion of the multiplications are *1010* and *1100* respectively. Then, by crossing bit by bit the two string we get $*11011000$.

small, the impact of performing broadcast is always heavy on energy consumption, especially in a sensor network. For this reasons in this paper we present a recovery procedure that does not uses broadcast at all, and we show that its performances are at least equal to it. We call this procedure *active recovery*. In this new solution, if a node n_i holding, among the others, the coordinate $*01001*$ discover that the node n_f with coordinate $*0100*$ is no longer available it will start a search for $*0100*$, the search is performed using a special type of message (recovery message) m_r. Each node that is crossed by a recovery message knows that each coordinate with prefix equal to the destination ($*0100*$ in this case) must not be used to evaluate the distance to it. This little variation tries to help the system getting out of the subtree generated by the node failure. The orphan node will try in sequence each of its neighbors to deliver the recovery message and the search will stop either when m_r is delivered or when all the neighbors have been contacted. The rest of the recovery procedure is the same as in [17]. Once the coordinate has been found, the holding node fixes the relationship with the affected node by giving it a new coordinate, in our case through n_4 and n_7.

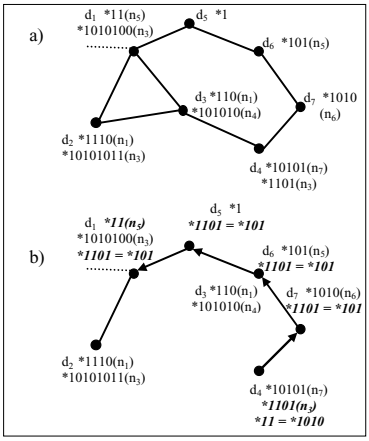


Fig. 2. Effects of Node Failure (d_3) during Routing of a Packet from Node d_1 to Node d_4

4 Lazy Recovery of Node Failures

W-Grid also include a *lazy recovery* feature. Besides active recovery we also let the nodes try to fix failures situations not solved by exploiting the traffic during queries routing.

Lazy recovery act as follows: whenever a node cannot recovery trough active recovery sets itself in a *recovery failed* state. When a node is in this state it will first of all notice all its neighbors about it and its neighbors will do the

same. Then, each node informed about this temporary state will add all of its coordinate to every query that it will be asked to route and that is evaluated to cross the node[2] in recovery failed state. The node in recovery failed state will scan each attached coordinate in the query message looking for a coordinate which is parent of the broken one, so that it can perform a recovery.

5 Local Learning

In this paper we improve the routing efficiency by introducing a new feature named local learning (LL). In a wireless environment uni-cast is never actually uni-cast, in fact, whenever a node communicates with one of its neighbors the communication is overheard by all of them, what it happens is that only the recipient of the communication will listen it. In the same way, each routing request exchanged among couples of nodes are heard by their respective neighbors. Our idea is that overhearing neighbors may not ignore the informations they "listen to" but they may process them instead, finding for help to give the (neighbor) routing nodes. It may happen that a node apparently farthest from the destination is aware of a node that would shorten the path, by giving back this information to its neighbor that was routing a message through another node it is possible that at the next routing the helping node will be chosen, and the path will be shorten. Local is referred to the fact that what nodes learn regards only their direct neighbors.

Every time that a packet p_i (data or query) with destination c_i is forwarded by a sensor d_f (forwarder) to a sensor d_r (receiver), each sensor that is within the radio range (neighbors N) of d_f is aware that the packet is being forwarded. As a consequence each sensor in N can discover (i) which virtual coordinate is the destination of p_i, (ii) which sensor d_r has been chosen towards that destination and (iii) which is the distance of p_i at d_r. Here comes the local learning. If any sensor in N, let us say d_l finds out that a neighbors, let us say d_{nf} with coordinate c_{nf} would have taken p_i closer than d_r then d_l temporarily stores the pair (d_{nf}, c_i) so that when it performs the next beaconing it informs d_f that a better path has been discovered. In this way, the next time that d_f needs to forward a packet to a destination whose prefix is c_i, d_l will be preferred to d_r. Figure 3 shows an example of local learning. Packet p_i with destination $*011$ must be routed by node d_f. By forwarding p_i to d_l the distance from d_f to the destination is 3 while by forwarding p_i to d_f the distance is 5. Local learning allows n_f to know that d_l is a better choice for routing packets whose destination is $*011\%$[3].

A possible variation of the strategy is to choose between d_l and d_r according to a certain probability, so that possible changes in network topology and consequently new possible paths can be caught even if some learning has already

[2] Each node can estimate if the query is likely to cross the orphan node by comparing query destination and node.

[3] % means a binary string of arbitrary length.

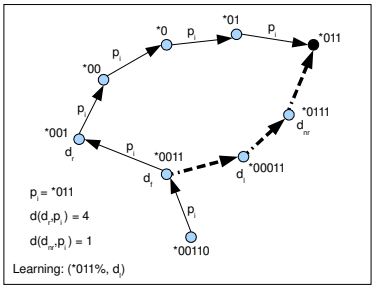


Fig. 3. Example of Local Learning at Node n_f

occurred. Simulation results show that the network gains in routing performances under these conditions.

6 Distances in the W-Grid Logical Network

We also added the Real Distance (RD) feature to W-Grid. Whenever a node n_j gets a coordinate from node n_i the new coordinate will be one bit longer that the father one. However n_i might have already split and while this information is known by n_j that will know about all the coordinates of it the same is not for n_j's neighbors which are not neighbors of n_i. Actually those neighbors could find useful such kind of information in order to get more precise distance values during routing. For this reasons routing table entry will also contain this integer value which represents the real distance among couple of nodes. In section 7 we evaluated network performance with respect to this feature.

7 Simulation Results

An extensive number of simulations have been conducted in order to evaluate the network routing performances, in term of average path length, and the robustness in W-Grid networks.

When comparing W-Grid with GPSR [12], which is the routing method employed by several existing solutions such as in [3] and in [20], it is appropriate to remind that in GPSR each sensor needs to be aware both of their physical location and the network perimeter. These constraints increase the cost of each sensor and limit GPSR usage possibility, for instance it cannot be used in indoor environments and in outdoor areas where the density of sensors is beyond the GPS precision, or when weather conditions are bad.

The simulation model consists of a square area $800 \times 800m$, in this area 205 nodes are randomly spread. Each sensor has its own ID and a radio range varying from $73m$ to $123m$ (ideal transmission) in order to get different densities, namely $4, 8$ and 12 neighbors per node respectively. For each scenario we ran 5

simulations and in each simulation we submitted 20000 queries to the system and then tested network robustness by turning off each node of the network one at a time. The simulator performed the following tasks:

- Random placement of sensors in a user-defined area;
- Generation of W-Grid coordinates at sensor exploiting implicit overhearing;
- Random generation of 20000 queries;
- Turning off of nodes at the delivery of queries, as previously described.

For each simulation run we observed:

- The variation in queries APL (Average Path Length), namely the number of hops necessary to resolve a query, between W-Grid with LL and RD and GPSR.
- The ratio of succeeded recovery in W-Grid scenarios;

7.1 Average Path Length Comparisons

Even if the comparison appears prohibitive, since GPSR can stay very close to the ideal routing algorithm by using physical position of nodes, W-Grid returns very good performances, especially considering that it does not require any kind of information about geographic position of nodes. Figures 4, 5 and 6 show that the number of hops (APL) is similar in W-Grid and GPSR especially when LL is applied. Besides, the flat look of the averages with respect with the number of coordinates shows that W-Grid behavior is stable according to that variable.

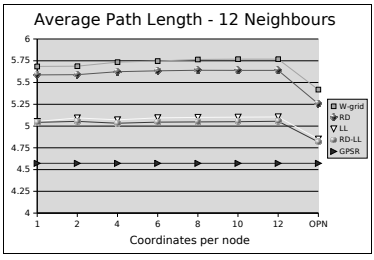


Fig. 4. Query APL in a Network with an Average of 12 Neighbors per Node

7.2 Recovery Failures

The second measure we evaluate is the ratio of failure recovery which is correctly performed according to the different node densities and the number of coordinates. We simulate two different recovery strategies:

- Active recovery;
- Lazy recovery.

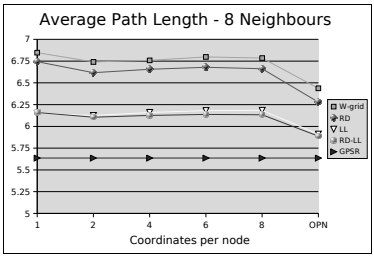


Fig. 5. Query APL in a Network with an Average of 8 Neighbors per Node

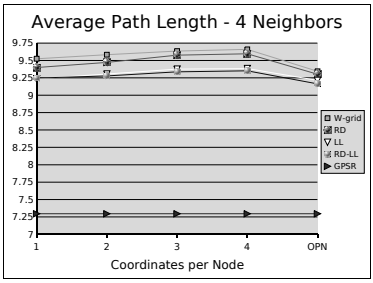


Fig. 6. Query APL in a Network with an Average of 4 Neighbors per Node

Where *lazy recovery* is performed whenever a node could not solve a failure situation with the active recovery. We present the results obtained with both strategies.

In each Figure 7, 8 and 9 we compare W-Grid efficiency with coordinates dependencies against the W-Grid solution exploiting message broadcast. Here, broadcast feature has been tried with different Time-To-Live (TTL) values and obviously the efficacy, not the efficiency, improves as TTL value increases. The fifth curve represents an unlimited broadcast which has been simulated whenever W-Grid could not be able to perform recovery. Figures show that almost every time that W-Grid was not able to perform recovery, unlimited broadcast was not able as well, meaning that preceding W-Grid version failed just because the network was partitioned due to device failure.

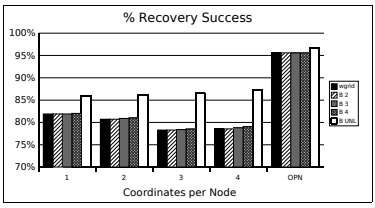


Fig. 7. Recovery Success Ratio with an Average of 4 Neighbors per Node

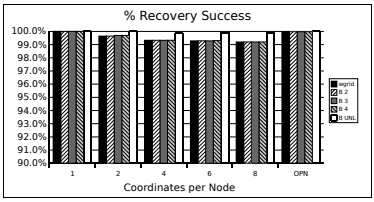


Fig. 8. Recovery Success Ratio with an Average of 8 Neighbors per Node

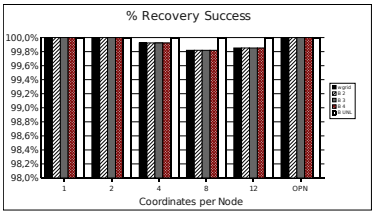


Fig. 9. Recovery Success Ratio with an Average of 12 Neighbors per Node

In Figure 10 we can see that the percentage of successful queries keeps really high even when the network is in an unstable state due to failed recoveries.

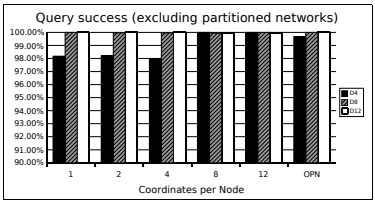


Fig. 10. Percentage of Successful Queries in the case of Recovery Failure

According to what shown by our experimental analysis (not included in this paper due to space limitations), the network traffic generated by W-Grid active failure recovery strategies compared with broadcast applied to W-Grid (with different level of broadcast propagations) expose an appreciable performance. Indeed, W-Grid heavily reduces the number of messages required for recovery.

8 Conclusions and Future Work

In this paper we presented W-Grid, a fault tolerant cross-layer infrastructure for routing and multi-dimensional data indexing and querying in ad-hoc sensor networks. After describing W-Grid we have introduced the new efficient recovery

capabilities and the local learning with a new distance evaluation technique, a routing technique that improves the average path length measures of W-Grid networks.

Besides, the work has showed that in case of failures, W-Grid guarantees network robustness while reducing the energy consumption with respect to existing solutions that instead require broadcast/flooding propagations. In particular, the simulations have measured both the efficacy and efficiency of the routing method and the recovery approach according to an extensive number of scenarios with different radio connectivity density, with several logical network topology changing the number of coordinates and with the treatment of node failures.

The results have highlighted that the routing performance of GPSR and W-Grid with local learning are quite similar despite GPSR requires GPS; finally the new recovery approach drastically reduces the network traffic while preserving the same recovery efficacy of solutions based on costly broadcast/flooding operations.

Future work is manly oriented towards integrating W-Grid with novel paradigms dictated by recent *Big Data* initiatives (e.g., [26,27]), perhaps inspired by approximation paradigms (e.g. [28,29]).

References

1. Monti, G., Moro, G., Lodi, S.: W*-Grid a robust decentralized cross-layer infrastructure for routing and multi-dimensional data management in wireless ad-hoc sensor networks. In: P2P 2007, pp. 159–166 (2007)
2. Monti, G., Moro, G.: Scalable multi-dimensional range queries and routing in datacentric sensor networks. In: Infoscale 2008(2008)
3. Li, X., Kim, Y., Govindan, R., Hong, W.: Multi-dimensional range queries in sensor networks. In: SenSys 2003, pp. 63–75. ACM Press, New York (2003)
4. Intanagonwiwat, C., Govindan, R., Estrin, D., Heidemann, J., Silva, F.: Directed diffusion for wireless sensor networking. IEEE/ACM Trans. Netw. 11, 2–16 (2003)
5. Ye, F., Luo, H., Cheng, J., Lu, S., Zhang, L.: A two-tier data dissemination model for large-scale wireless sensor networks. In: MobiCom 2002, pp. 148–159. ACM Press, New York (2002)
6. Perkins, C., Bhagwat, P.: Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In: SIGCOMM 1994, pp. 234–244. ACM Press, New York (1994)
7. Chiang, W.L.C., Wu, H., Gerla, M.: Routing in clustered multihop, mobile wireless networks. In: IEEE SICON, pp. 197–211 (1997)
8. Murthy, S., Garcia-Luna-Aceves, J.J.: An efficient routing protocol for wireless networks. Mob. Netw. Appl. 1, 183–197 (1996)
9. Perkins, C., Royer, E.: Ad-hoc on-demand distance vector routing. In: WMCSA 1999, p. 90. IEEE Computer Society (1999)
10. Johnson, D., Maltz, D., Broch, J.: DSR: the dynamic source routing protocol for multihop wireless ad hoc networks. Ad hoc Networking, 139–172 (2001)
11. Park, V., Corson, M.S.: A highly adaptive distributed routing algorithm for mobile wireless networks. In: INFOCOM 1997, p. 1405. IEEE Computer Society, Washington, DC (1997)

12. Karp, B., Kung, H.: GPSR: greedy perimeter stateless routing for wireless networks. In: MobiCom 2000, pp. 243–254. ACM Press (2000)
13. Kuhn, F., Wattenhofer, R., Zhang, Y., Zollinger, A.: Geometric ad-hoc routing: of theory and practice. In: PODC 2003, pp. 63–72. ACM Press (2003)
14. Rao, A., Papadimitriou, C., Shenker, S., Stoica, I.: Geographic routing without location information. In: MobiCom 2003, pp. 96–108. ACM Press (2003)
15. Moscibroda, T., O'Dell, R., Wattenhofer, M., Wattenhofer, R.: Virtual coordinates for ad hoc and sensor networks. In: DIALM-POMC 2004, pp. 8–16. ACM Press, New York (2004)
16. Bischoff, R., Wattenhofer, R.: Analyzing connectivity-based multi-hop ad-hoc positioning. In: PERCOM 2004, p. 165. IEEE Computer Society, Washington, DC (2004)
17. Moro, G., Monti, G.: W-Grid: a self-organizing infrastructure for multi-dimensional querying and routing in wireless ad-hoc networks. In: P2P 2006, pp. 210–220 (2006)
18. Ratnasamy, S., Karp, B., Shenker, S., Estrin, D., Govindan, R., Yin, L., Yu, F.: Data-centric storage in sensornets with ght, a geographic hash table. Mob. Netw. Appl. 8, 427–442 (2003)
19. Greenstein, B., Estrin, D., Govindan, R., Ratnasamy, S., Shenker, S.: Difs: A distributed index for features in sensor networks. In: IEEE WSNA 2003, pp. 163–173. IEEE Computer Society (2003)
20. Xiao, L., Ouksel, A.: Tolerance of localization imprecision in efficiently managing mobile sensor databases. In: MobiDE 2005, pp. 25–32. ACM Press, New York (2005)
21. Eriksson, J., Faloutsos, M., Krishnamurthy, S.: Peernet: Pushing peer-to-peer down the stack, in: IPTPS. In: Kaashoek, M.F., Stoica, I. (eds.) IPTPS 2003. LNCS, vol. 2735, pp. 268–277. Springer, Heidelberg (2003)
22. Monti, G., Moro, G., Sartori, C.: WR-Grid: A scalable cross-layer infrastructure for routing, multi-dimensional data management and replication in wireless sensor networks. In: Min, G., Di Martino, B., Yang, L.T., Guo, M., Rünger, G. (eds.) ISPA Workshops 2006. LNCS, vol. 4331, pp. 377–386. Springer, Heidelberg (2006)
23. Monti, G., Moro, G.: Self-organization and local learning methods for improving the applicability and efficiency of data-centric sensor networks. In: Bartolini, N., Nikoletseas, S., Sinha, P., Cardellini, V., Mahanti, A. (eds.) QShine 2009. LNICST, vol. 22, pp. 627–643. Springer, Heidelberg (2009)
24. Ouksel, A.M., Moro, G.: G-grid: A class of scalable and self-organizing data structures for multi-dimensional querying and content routing in P2P networks. In: Moro, G., Sartori, C., Singh, M.P. (eds.) AP2PC 2003. LNCS (LNAI), vol. 2872, pp. 123–137. Springer, Heidelberg (2004)
25. Moro, G., Ouksel, A., Litwin, W.: GGF: A Generalized Grid File for Distributed Environments, Technical Report, DEIS Univ. of Bologna, Univ. of Illinois at Chicago (2002)
26. Cuzzocrea, A., Saccà, D., Ullman, J.: Big Data: A Research Agenda. In: ACM IDEAS 2013, pp. 198–203 (2013)
27. Cuzzocrea, A., Song, I.-Y., Davis, K.C.: Analytics over Large-Scale Multidimensional Data: The Big Data Revolution! In: ACM DOLAP 2011, pp. 101–104 (2011)
28. Cuzzocrea, A., Furfaro, F., Mazzeo, G.M., Saccà, D.: A Grid Framework for Approximate Aggregate Query Answering on Summarized Sensor Network Readings. In: OTM Workshops, pp. 144–153 (2004)
29. Cuzzocrea, A., Chakravarthy, S.: Event-based lossy compression for effective and efficient OLAP over data streams. Data Knowl. Eng. 69(7), 678–708 (2010)

Author Index