# NoXperanto: Crowdsourced Polyglot Persistence

Antonio Maccioni, Orlando Cassano, Yongming Luo, Juan Castrejón, Genoveva Vargas-Solar

# NoXperanto: Crowdsourced Polyglot Persistence

Antonio Maccioni, Orlando Cassano, Yongming Luo, Juan Castrejón, and Genoveva Vargas-Solar

***Abstract*—This paper proposes NoXperanto, a novel crowdsourcing approach to address querying over data collections managed by polyglot persistence settings. The main contribution of NoXperanto is the ability to solve complex queries involving different data stores by exploiting queries from expert users (i.e. a crowd of database administrators, data engineers, domain experts, etc.), assuming that these users can submit meaningful queries. NoXperanto exploits the results of "meaningful queries" in order to facilitate the forthcoming query answering processes. In particular, queries results are used to: (i) help non-expert users in using the multi-database environment and (ii) improve performances of the multi-database environment, which not only uses disk and memory resources, but heavily rely on network bandwidth. NoXperanto employs a layer to keep track of the information produced by the crowd modeled as a Property Graph and managed in a Graph Database Management System (GDBMS).**

***Index Terms*—Polyglot persistence, crowdsourcing, multi-databases, big data, property graph, graph databases.**

## I. Introduction

BIG datasets are not uniform collections of homogeneous data and, for this reason, they are not logically stored in one single database. In addition, the size of big datasets is such that it is not possible to adopt data cleaning, dataset preparation and integration using classic methods. Otherwise, data would never be ready for being analyzed and exploited in reasonable time.

Often, these datasets are sharded across distributed persistence supports that adopt different access and management policies, different degrees of availability, fault tolerance and consistency.

*Polyglot Persistence* [1] is a brand new term to indicate the combination of approaches that cope with a variety of persistence systems and that are used for "integrating" heterogeneous datasets.

As part of the emerging polyglot persistence movement [1], the simultaneous use of multiple SQL, NoSQL and NewSQL data stores is gradually becoming a common practice in modern application development [2], [3]. They enable the integration of these data stores for managing big datasets in a scalable and loosely coupled way. This approach seems to be a pertinent strategy to deal with big datasets integration. Nonetheless, the combination of these heterogeneous data stores, flexible schemas and non-standard APIs, represent an added complexity for application developers. For instance, considering that data are spread across multiple data stores, each of which possibly relies on distinct data models (graph, document, etc.), developers must be familiar with a high number of implementation details, in order to effectively work with and maintain the overall data entities.

Providing an integrated view of the underlying data collections for enabling querying, is still an open issue, often solved at the application code level. Given that there is no uniformization of schemas and there is a lack of meta-information available about the data (i.e. mappings, meta-data, semantic equivalences, data similarities), query results are not integrated collections of data, often, they are bags of non related data collections. It follows that, the quality of querying in such polyglot systems is poor.

**Contribution.** This paper proposes NoXperanto, a novel crowdsourcing[1] approach to address querying over data collections managed by polyglot persistence settings.

We avoid the pursue of uniformity, but rather, we preserve the variety of original systems and languages. The main contribution of NoXperanto is the ability to solve complex queries involving different data stores by exploiting queries from expert users (i.e. a crowd of database administrators, data engineers, domain experts, etc.), assuming that these users can submit meaningful queries.

Crowds have been revealed to be effective to solve queries over a single database [4], but they have never been used to query a multi-database system, where the required expertise (about schemas, models, data formats and instances of the database) cannot be superficial. In fact, from the structure of the queries, we can infer who is expert.

NoXperanto exploits the results of such "meaningful queries" in order to facilitate the forthcoming query answering processes. In particular, the results of queries are used to: (i) help non-expert users in using the multi-database environment and (ii) improve performances of the multi-database environment, which not only uses disk and memory resources, but heavily rely on network bandwidth.

[1]The modality to fulfil a target by relying on the contributions coming from a large group of people.

Manuscript received on May 3, 2014; accepted for publication on June 10, 2014, published on November 15, 2014.

Antonio Maccioni is with the Università Roma Tre, Rome, Italy (e-mail: maccioni@dia.uniroma3.it).

Orlando Cassano is with the Université Libre de Bruxelles, Brussels, Belgium (e-mail: orlando.cassano@cetic.be).

Yongming Luo is with the Eindhoven University of Technology, the Netherlands (e-mail: y.luo@tue.nl).

Juan Castrejón is with the Université de Grenoble, Grenoble, France (e-mail: juan.castrejon@imag.fr).

Genoveva Vargas-Solar (corresponding author) is with the Centre national de la recherche scientifique (CNRS), LIG-LAFMIA labs, Saint Martin d'Hères, France (e-mail: genoveva.vargas-solar@imag.fr).

NoXperanto employs a *layer* to keep track of the information produced by the crowd. This is modeled as a Property Graph and thus, it is persisted by a Graph Database Management System (GDBMS).

**Outline.** The remainder of the paper is organized as follows. While Section III introduces the concepts that we need throughout the paper, Section II discusses works that are related to our. Section IV explains our approach in detail. Section V describes implementation issues. Section VI concludes the paper with future perspectives of research.

## II. Related Works

Most information systems are supported by distributed and heterogeneous data sources: multi-databases, data warehouses and Web portals. Such systems are, in fact, mediation infrastructures that emerged more than 20 years ago to enable transparent access to multiple data sources through querying, analysis, navigation and management facilities.

With the advent of NoSQL stores and polyglot persistence approaches, problems of data integration emerge with some specifities: the high heterogeneity of data models (e.g., NoSQL underlying data models are not standardized), the absence of semantic descriptions, the volume of data to integrate, just to name a few. Transformation and other well-known techniques to bridge different databases seem hard to employ at large scale [5]; they would bring data duplication that is unfeasible when the data size is huge. In order to deal with heterogeneity, the majority of information systems define an integrated schema. However, the integration process, generally controlled by the administrator of the system, is hardly suited for managing data in highly distributed and evolving environments such as the Web. Hardly coupled multi-sources storing huge datasets with semantic information (e.g. global schemas, mappings, rewritting rules) can be restrictive and expensive for applications that require simplicity, performance and large-scale dynamicity.

Advances in polyglot persistence querying are addressing the integration of heterogeneous data by providing pivot querying languages like UnQL that provide some operators for integrating graph, document, key value and relational data. There exists equivalent works which consider a unified language, proposing some SQL extension [6], [7], [8], [9], [10], or consider a metamodel approache such as the Apache's MetaModel project[2].

To overcome existing problems of data integration, we propose a solution to exploit a crowd of experts in a transparent way. Experts are all the users who are able to submit meaningful queries. We do not assume having a unified query language, but rather we follow the idea of the polyglot persistence movement to support multiple languages.

The idea of using crowdsourcing for answering queries over a database has been covered in previous works [7], [8], [11], [9], [4], [10], [12]. In particular, CrowdDB [4] integrates human inputs within a query plan that contains operations that usual database systems cannot always answer correctly (i.e. matching, ranking, etc.). RandomDB [12] answers non deterministic queries using social network crowds with the aim to replace existing random number generators.

[2]http://metamodel.eobjects.org/

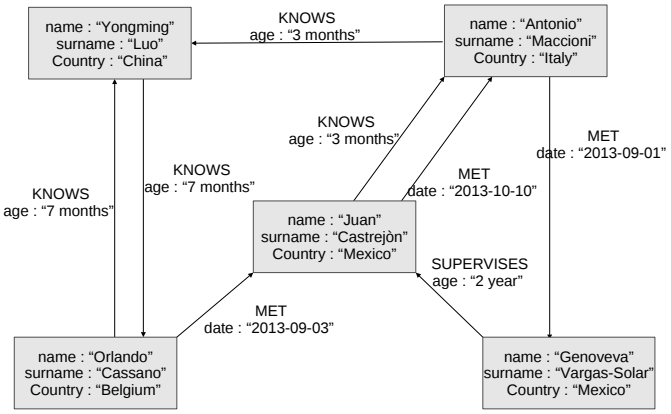


Fig. 1. An example of property graph.

## III. Querying a polyglot multi-store

This section introduces basic concepts that are needed to introduce our approach in the next section.

**Property Graph.** Graph data models are able to generalize other data models:

- *relational data:* both schema and instance of relational data can be modeled with graphs [13];
- *XML data:* XML document is modeled as a tree and a tree is an acyclic, undirected graph;
- *document stores:* each document consists of nested key-value pairs (therefore a tree), so a document store can be modeled as a set of trees;
- *key-value stores:* they corresponds to a set of nodes of the property graph.

NoXperanto does not generalize, nor integrates, other models; rather, it adopts a graph data model in order to link more easily the parts of the entities that are spread across different databases. In particular, we use the *property graph* model [14]. Briefly, a property graph is a multi-graph (a graph where two nodes can be connected by more than one edge) where every node and every edge has associated a set of *properties*. A property is a key-value pair, denoted in the format of $< key, value >$.

An instance of property graph is reported in Figure 1. This represents people (i.e. the nodes) and their relationships. For example the node representing a person whose name is *Genoveva* supervises the node representing the person whose

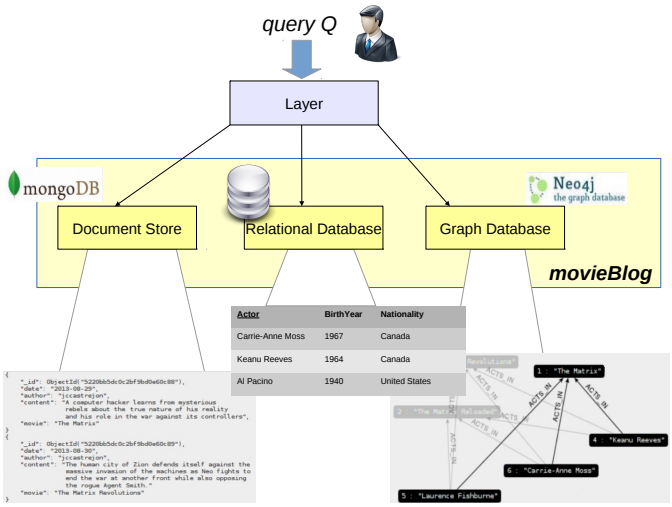


Fig. 2. The movie blog environment.

name is *Juan*. In this case *¡name : Genoveva¿* is a property where *name* is the key and *Genoveva* is the value.

Though the property graph is directed, we simplify it using the undirected version. The property graph is the "de-facto" standard model adopted by the GDBMSs (i.e. Neo4J, OrientDB, etc.). We will see in the Section V more details about it, since we make use of a GDBMS.

**Running Example.** To facilitate the understanding of the approach, let us consider the following running example. We have an infrastructure of databases, called *movieBlog*, behind a big blog website talking about movies. Information about entities such as the movies are splitted into different databases. The infrastructure consists in a multi-database, where we have a document store containing blog entries, a relational database storing information about actors, and a graph database keeping track of the movies' cast. Figure 2 depicts our scenario.

The definition of the databases of our running example are in Figure 3.

```
[DEFINITIONS]
define "blogEntries" as document
define "movies" as graph
define "actors" as table
```

Fig. 3. The database definition of the databases in Figure 2.

The example defines three databases of different kind: a relational database (i.e. table *actors*), a graph database (i.e. graph *movies*) and a document store (i.e. document *blogEntries*). The keys of these databases are the attributes *Actor*, *id* and *_id*, respectively for the databases *actors*, *movies* and *blogEntries*.

**Polyglot Environment.** Databases contain *entities* (e.g., a tuple in a relational database or a document in a document store). Every entity can be referred with a *key* (e.g., a primary key in a relational database or a document id in a document store). Queries are expressions that aim at retrieving: (i) an entity, (ii) a part of an entity or (iii) a combination of entities. Without referring to a particular language or algebra, we can refer to these tasks with general operator of *selection* ($\sigma$), *projection* ($\pi$) and *join* ($\bowtie$), respectively.

A $\sigma$ query over a polyglot environment still involves separate answering processes, where database instances are independent from each other. In this case, the query can be dispatched to all the underlying databases, after an eventual syntactic translation of the query expression. A $\pi$ is straightforward since it operates locally by extracting a sub-part of a result. While the semantics of $\sigma$ and $\pi$ in a polyglot environment are well-defined, the semantics of the $\bowtie$ needs to be clarified. In fact, if we join heterogeneous databases we can hardly rely upon a single algebra. We define such semantics as follows. Let us suppose to have two entities $e_i$ and $e_j$ belonging, respectively, to two heterogeneous databases. Among the others, the two entities have attributes $a_i$ and $a_j$, respectively. Let us also suppose that there exists a comparing operator $\diamond$. A join $e_i \bowtie_{e_i.a_i \diamond e_j.a_j} e_j$ is the union between the information content of $e_i$ and the content of $e_j$ iff the predicate $e_i.a_i \diamond e_j.a_j$ is satisfied.

Clearly, we can develop a join operator in a polyglot environment at a code-level exploiting existing join algorithms (e.g., nested-loop style join). In this context, such operations are very expensive as we cannot rely upon the optimizations of the DBMSs but we are compelled to load all the data in memory and, sometimes, to transfer them over a network.

Next section explains how NoXperanto is able to answer queries using a crowd of experts and, in particular, it will focus on how the join operators can be computed efficiently.

## IV. NoXperanto

This section explains our approach in detail. We first give an overview of the overall approach. Then, we conclude the section describing some use case in order to point out the advantages of the system.

### A. Crowdsourcing Approach

In NoXperanto we aim at solving complex queries over a system containing several heterogeneus databases. To perform such queries we have to keep track of the relationships among entities stored in different databases. We employ two ways to indicate these relationships: one is *explicit* and the other is *implicit*.

**Explicit Working Mode.** In the *explicit* manner, the user can define how two classes of entities in different databases are related. For example, in Figure 4 we define that an entity of the database *blogEntries* is the same of an entity in the database *movies* if the value of *blogEntries.movie* is equal to the value of *movies.titles*.

```
[DEFINITIONS]
define "comments"
  on "blogEntries.movie" = "movie.titles"
  as link
```

Fig. 4. The explicit definition of relationships.

In this case we can exploit several techniques (i.e. ontology matching, schema matching, string matching, etc.) to find the instances of such definitions. They work at schema level and are very expensive to be performed at run-time. We can mitigate this complexity by using an hybrid approach between this explicit mode with the implicit mode that is explained next, but this lies outside the scope of this paper.

However, in many cases the administrator does not explicitly specify such definitions. It turns out that an automatic discovery of the relationships might be very useful. NoXperanto provides such a mechanism through the *implicit* modality.

**Implicit Working Mode.** The *implicit* working mode is driven by a crowdsourcing approach. Crowdsourcing is employed in different contexts to solve difficult problems and to build up knowledge bases relying on the effort of many people. Usually, a problem is split into many sub-problems that are solved through the so called microtasks. A microtask is a complicated task for a computer but it is easy for a person. The final problem is solved by considering all the microtasks that people of the crowd have processed. Often, these people are unaware of the problem or do not even know that they are processing a microtask.

The *implicit* working mode expects the system to be able to recognize the relationships. In this case, the system uses the knowledge of a crowd of experts, which appears when they submit complex queries. If those queries are also meaningful, in the sense that they produce a non empty set of results, we persist the relationships between entities of the two (or more) databases in our *property graph layer*.

More in detail, we extract the predicate of the join conditions from such queries to define a *crowd link*. For example, let us imagine an expert submitting $\sigma_{id,Nationality}$ $(actors \bowtie_{Actor=id} movie)$. In a SQL-like language, the query will look like the following:

```
SELECT id, Nationality
FROM actors, movies
WHERE actors.Actor = movies.id
```

The result of the query is {*¡Keanue Reeves, Canada¿, ¡Carrie-Anne Moss, Canada¿*}, thus, it is a non empty result set and the query is meaningful. In the answering process we have identified some relationships between entities in *movies* and entities in *actors*. For each of them, NoXperanto persists a crowd link on the property graph layer as in Figure 5.

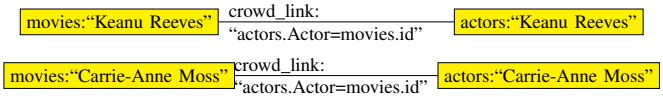


Fig. 5. Crowd links explicitly produced.

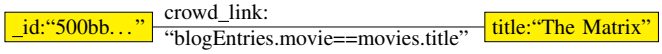


Fig. 6. Crowd links produced in Use case 2.

### B. The Approach at Work

In this section we describe a sequence of four use cases to show the real benefits of a crowdsourcing approach in this context. The cases alternate queries from a non expert and expert users. For the sake of simplicity we consider a SQL-like syntax but of course, it is just the semantics of the language and does not refer to any relational database. We refer the multi-database managed by NoXperanto with *movieBlog*. In this way, we query our datasets in a transparent way. The four cases are defined as follows:

– *Use case 1*: a non-expert user asks for all information about “The Matrix”. It is a simple keyword search-like query (either submitted by a human being or by a programmatic access), so we cannot assume that the user is an expert.

```
SELECT *
FROM movieBlog
WHERE movieBlog.title == ``The Matrix''
```

The result of the query is

```
{year=1999}
```

since we have found an attribute *title* within the databases in *movieBlog* and an entity where the value of such attribute is “The Matrix”.

– *Use case 2*: a domain expert perform the following query:

```
SELECT *
FROM movies
JOIN blogEntries
  ON blogEntries.movie == movies.title
```

Note that the expert does not generally refer to *movieBlog* but to a precise database. The result of the query is:

```
{year=1999,
content="A computer hacker learns from
         mysterious rebels about the
         true nature...''
author="jccastrejon''}
```

We can say that there is a relationship between *blogEntries.movie* and *movies.title*. Since a non-expert user could not write such a join with meaningful

results, we determine that this user was an expert user. Consequently, she can be included within our crowd. In fact, the system stores a crowd link for each of the results as in Figure 6. We have bridged the document store with the graph databases at runtime.

– *Use case 3*: a non-expert user submits the query of Use case 1, but in this case the result is different.

```
{year=1999,
content="A computer hacker learns from
         mysterious rebels about the
         true nature...''
author="jccastrejon''}
```

We provided information to the user that we were not able to retrieve before. This additional information is provided by answering a join-free query. This is due to the information within the *layer*, which allows to bridge sharded parts of the same entity.

– *Use case 4*: another expert user submits the query of Use case 2. The scenario is similar to use case 2, but the query answering is much more efficient. We do not have to perform an expensive join operation. We can directly exploit the presence of the crowd links to determine a pairs of entities to form the final results.

## V. Implementation Issues

This section provides and overview of the implementation concerns that are required to develop the NoXperanto approach based on the requirements outlined in the previous sections.

**Data Layer Issue.** We manage the heterogeneity of multiple data stores by relying on a data layer based on the property graph data model (see Section III). We implemented this layer on an emerging GDBMS, that is Neo4J.[3] It provides a REST interface so that the interaction with the applications running in the polyglot environment is facilitated. Moreover, this interface would provide operations to manage data entities and links between them. For example, the specific syntactic sugar to specify when to consider the crowd links in the query answering.

**Language Issue.** Our approach does not consider a unified query language for polyglot persistence applications, but rather rely on the existing language support provided by scalable data stores. Thus, we propose to provide extensions for each of these languages, based on the general query semantics described in Section III. In particular, these extensions rely on the REST interface of the property graph model to manage the link and join operations described in Section IV.

To implement these language extensions we intend to follow a model-driven approach, as proposed in our previous work.[4] In particular, the language definitions would be implemented using the Xtext framework [15], while the implementation of the link and join operations would be conducted with the Acceleo project,[5] by relying on text templates that refer to the graph model REST interface. The native query mechanisms of each of the supported data stores would be used to retrieve the data in each of the systems.

**Consistency Issue.** Our approach assumes an eventual consistency model [16] in the persistence of the crowd links among the distributed entities. As a consequence, even when a link between data entities has been identified, users executing the same query may not always receive the same result. Nonetheless, each of the data stores in the environment has its consistency semantics.

To handle this heterogeneity in the consistency semantics we propose to extend our language support with operators to allow the user to specify the level of consistency that he expects from each of the supported data stores. We intend to implement this functionality based on the operators that few systems (e.g., Riak,[6] a key-value data store) already provide to trade availability for consistency on a per-request basis.

**Crowd Management Issue.** We have developed a small utility to manage the crowd of experts. This consists, basically, on a query parser and a small interface where the administrator can check the current state of the crowd links.

## VI. Conclusions and Perspectives

In this paper we have presented NoXperanto, an approach to solve queries over an heterogeneus environment of databases using the knowledge of a crowd of experts. This knowledge is extracted from the results of the queries. In NoXperanto we avoid expensive pre-processing. As a result, we are able to scale with respect to the number of the databases within our environment.

Our future work will be devoted to finish the system, implementing an interface for setting up the multi-database environment and allowing the users to specify weather or not the system has to use the crowd links. The approach opens several research direction. In particular, we will investigate other scenarios where the results of a query can be exploited to facilitate forthcoming query answering.

## Acknowledgments

The ideas within this paper were developed during the EDBT Summer School 2013 in Aussois (France). The authors of this paper are grateful to the EDBT association for the organization of the Summer School and to all the speakers for their helpful suggestions.

[3] http://www.neo4j.org/
[4] https://github.com/jccastrejon/edbt-unql/
[5] http://www.eclipse.org/acceleo/
[6] http://basho.com/riak/

## REFERENCES


[1] P. J. Sadalage and M. Fowler, *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Addison-Wesley, 2012.

[2] B. F. Cooper, "Spanner: Google's globally-distributed database," in *Proceedings of the 6th International Systems and Storage Conference*. ACM, 2013, p. 9.

[3] D. Borthakur, "Petabyte scale databases and storage systems at facebook," in *SIGMOD Conference*, 2013, pp. 1267–1268.

[4] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin, "CrowdDB: answering queries with crowdsourcing," in *SIGMOD Conference*, 2011, pp. 61–72.

[5] R. De Virgilio, A. Maccioni, and R. Torlone, "Converting relational to graph databases," in *GRADES*, 2013.

[6] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller, "Human-powered sorts and joins," *Proc. VLDB Endow.*, vol. 5, no. 1, pp. 13–24, Sep. 2011.

[7] J. Selke, C. Lofi, and W.-T. Balke, "Pushing the boundaries of crowd-enabled databases with query-driven schema expansion," *Proc. VLDB Endow.*, vol. 5, no. 6, pp. 538–549, Feb. 2012.

[8] A. Bozzon, M. Brambilla, and S. Ceri, "Answering search queries with CrowdSearcher," in *Proceedings of the 21st international conference on World Wide Web*, ser. WWW'12, 2012, pp. 1009–1018.

[9] G. Demartini, B. Trushkowsky, T. Kraska, and M. J. Franklin, "CrowdQ: Crowdsourced query understanding," in *CIDR*, 2013.

[10] H. Park, R. Pang, A. Parameswaran, H. Garcia-Molina, N. Polyzotis, and J. Widom, "An overview of the deco system: data model and query language; query processing and optimization," *SIGMOD Rec.*, vol. 41, no. 4, pp. 22–27, jan 2013.

[11] J. Wang, T. Kraska, M. J. Franklin, and J. Feng, "CrowdER: crowdsourcing entity resolution," *Proc. VLDB Endow.*, vol. 5, no. 11, pp. 1483–1494, Jul. 2012.

[12] R. De Virgilio and A. Maccioni, "Generation of reliable randomness via social phenomena," in *MEDI*, 2013, pp. 65–77.

[13] R. Angles and C. Gutiérrez, "Survey of graph database models," *ACM Comput. Surv.*, vol. 40, no. 1, 2008.

[14] M. A. Rodriguez and P. Neubauer, "Constructions from dots and lines," *CoRR*, vol. abs/1006.2361, 2010.

[15] M. Eysholdt and H. Behrens, "Xtext: implement your language faster than the quick and dirty way," in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, ser. SPLASH'10. New York, NY, USA: ACM, 2010, pp. 307–309. [Online]. Available: http://doi.acm.org/10.1145/1869542.1869625

[16] D. Pritchett, "BASE: An Acid alternative," *Queue*, vol. 6, no. 3, pp. 48–55, May 2008. [Online]. Available: http://doi.acm.org/10.1145/1394127.1394128