TransactiveDB: Tapping into Collective Human Memories

Michele Catasta†, Alberto Tonon*, Djellel Eddine Difallah*, Gianluca Demartini*, Karl Aberer†, and Philippe Cudre-Mauroux*
†EPFL—Switzerland *eXascale Infolab, University of Fribourg—Switzerland
(firstname.lastname)@epfl.ch (firstname.lastname)@unifr.ch

ABSTRACT

Database Management Systems (DBMSs) have been rapidly evolving in the recent years, exploring ways to store multi-structured data or to involve human processes during query execution. In this paper, we outline a future avenue for DBMSs supporting transactive memory queries that can only be answered by a collection of individuals connected through a given interaction graph. We present TransactiveDB and its ecosystem, which allow users to pose queries in order to reconstruct collective human memories. We describe a set of new transactive operators including TUnion, TFill, TJoin, and TProjection. We also describe how TransactiveDB leverages transactive operators—by mixing query execution, social network analysis and human computation—in order to effectively and efficiently tap into the memories of all targeted users.

1. INTRODUCTION

Humans can store, process—and eventually forget—personal memories on their own, but also collectively. The notion of *transactive memory* [6] was established almost 30 years ago to denote the capacity of groups of individuals to collectively store and retrieve knowledge. Classical examples of transactive systems are older couples or families living together. Even if an individual in the group cannot remember a specific fact, he/she will often have a systematic way of retrieving the desired piece of information (e.g., by asking a specific individual from the group).

Recently, crowdsourcing has been used to complement classical database systems by leveraging human intelligence at scale [3]. While crowdsourcing DB systems can answer queries that purely automated systems cannot, they are still for the most part confined to relatively simple and generic tasks such as translating sentences or finding information on the Web.

In this paper, we present our vision for TransactiveDB: a futuristic system leveraging personal memories collectively and automatically in order to answer queries whose results are not necessarily available in digital form, but are rather in the minds of relevant individuals. In Hippocampus [1], we presented a first example of transactive search, which was able to reconstruct a shared memory more effectively than either machine-based or crowdsourcing-based approaches. In the following, we outline a generic architecture for a complete transactive DB system combining distributed query execution, human intelligence, and social interactions to satisfy a novel class of information needs.

TransactiveDB works as a Peer Data Management System [4] in which each user represents an autonomous system combining relational tables (i.e., digital information) and human memories (i.e., personal information). Our system targets the union of digital and personal information across the network of nodes. Both sources of information are fundamental to answer transactive queries expressing information needs that can only be satisfied by tapping into certain group memories, e.g., "Who is the person on the left of this picture that I took during the eXascale lab retreat in Anzere, Switzerland on Jan 30th, 2014?"

The rest of this paper is structured as follows. We start below by describing the rational behind our system. Section 3 presents an overview of TransactiveDB's architecture. We describe the social graph supporting TransactiveDB's query execution in section 4 and the new set of transactive operators that we introduce in section 5. Finally, we lay out a research agenda for TransactiveDB in Section 6.

2. SYSTEM RATIONALE

> "What was the name of that amazing drink I ordered yesterday at the hipster bar?"

There are many reasons why a person might wish to ask such a question; more importantly in our context, not everybody can provide an answer to it. That question actually translates into a transactive query that can neither be answered by querying the web, nor by asking arbitrary Internet users via crowdsourcing. In order to give a precise answer to that query, one has to have interacted with the requester in the context of that query, meaning that one needs to have been with him at the hipster bar (direct interaction) or, at least, have heard about that event (indirect interaction). This question is an example of a *transactive fill query*, a particular kind of transactive query supported by TransactiveDB that is processed iteratively and collaboratively by exploiting the interaction graph, i.e, the graph representing the relations among all people involved in the event the query refers to (a more detailed description of the interaction graph is given below in Section 4). In that case, the selection query targets one particular attribute (the name of a drink), but it might

Proceedings of the VLDB Endowment, Vol. 7, No. 14

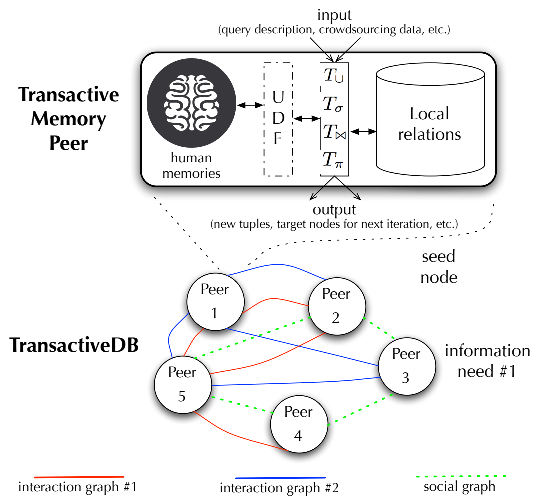


Figure 1: The architecture of TransactiveDB. In the depicted scenario, Peer_3 has an information need (#1), which can be satisfied by the Transactive Memory peers belonging to the interaction graph #1. As such, Peer_3 exploits its social network connections to discover that Peer_2 can behave as the seed node for the query (i.e., Peer_2 can route the query to all the other peers involved in the relevant interaction graph).

also involve a second transactive element in order to determine the selection condition (i.e., to determine the name of the hipster bar).

TransactiveDB is also able to process further types of transactive queries such as transactive joins for matching entities, or transactive unions for completing missing data (see below Section 5). TransactiveDB could also be leveraged to reconstruct the transactive memory of an enterprise. In that case, employees could for example programmatically tap into the memories of their colleagues who participated in a given meeting in order to recover detailed events or decisions taken during the meeting.

A transactive DBMS sets itself apart from a crowd-powered DBMS in the way participants are selected and incentivised, as well as in the way queries are collectively and iteratively executed. Where crowdsourcing employs a large number of anonymous, paid workers, our system leverages the acquaintances of the query requester to iteratively augment a shared corpus of information, i.e., the collective memory relating to the query. Subqueries are dispatched to the most suitable individuals depending on the interaction context (extending in that sense our previous push-crowdsourcing approach [2]). Nonetheless, we leverage classical crowdsourcing techniques to execute generic operators that do not require any specific knowledge about the requester, e.g., "Are these two photos depicting the same person?".

3. OVERALL ARCHITECTURE

We envision a decentralized data management system that exposes memories of individuals or groups in order to answer transactive queries. A simplified architecture of TransactiveDB is depicted in Figure 1. The first key component of our system is the Transactive Memory Peer (a node hereafter), mainly composed of the memory of a person and a physical store onto which particular memories gets transcribed over time.

The physical store is a classical database system running for example in the cloud or on a personal device of the user. All the DBMS components are *transactive-enabled*, in the sense that they can execute transactive queries; in addition, these systems can also query the crowd via generic crowd operators (such as those defined in CrowdDB [3]). The memory transcription process can be triggered by two different types of events i) a voluntary act of documenting an experience, e.g., "record the following grocery list", or ii) the reception of a query coming from the user or from a different node e.g.: "How much did last night meal at the restaurant cost?".

The transcribed data is organized into relational tables that users create following a standard schema definition. With the proper security and privacy mechanisms (e.g., access control credentials), the data can be exposed and queried by trusted nodes in the system.

The second key element of our architecture is the *interaction graph* connecting the different nodes participating in query execution. The interaction graph is a subset of the underlying social network connecting the different end-users. The exact set of nodes and edges constituting the interaction graph is progressively elicited as the transactive query gets executed. Different queries can hence generate different interaction subgraphs (see Section 4).

Finally, query execution in TransactiveDB is iterative, as it runs until a convergence criterion has been reached or the budget used to operate the transactive memory system is exhausted. At each step, the transactive memory peers are first selected and contacted based on the information contained in the interaction graph. Each Transactive Peer that the system reaches tries to retrieve the requested data using its local store. In case the information retrieved is deemed acceptable, the data is directly returned to the requester. Otherwise, the system generates a human-readable representation of the query, to which the local user can answer by filling out a Web-based form with information from his/her memory. At this step, missing data values can also be gathered trough conventional crowdsourcing (e.g., emails of newly discovered participants). Finally, the results are post-processed and merged together and the peers selected for the following iteration are selected.

4. LEVERAGING HUMAN MEMORIES THROUGH INTERACTIONS

TransactiveDB aims at reconstructing and gathering distributed information stored as human memories, be it digitally documented or dwelling inside the individuals' brains. Reconstructing such memories *a posteriori* requires some *a priori* exposure to the required information as well as explicit social interactions to recompose the missing pieces. Social networks are today the canonical way of representing social interactions in a digital form. In fact, the existence of multiple social networks (e.g., professional, friendship, or celebrity networks) is a manifestation of the diverse social interactions humans nurture. We define a *social graph* as the implicit or explicit graph encompassing the different interactions people have. Since a transactive query typically requires some very specific knowledge, it is only aimed at a subset of the *social graph*, called the interaction graph (see below and Figure 2).

4.1 Interaction Graph

In our setting, collective memories are associated to a specific context where participants interact with each other, forming what we call an *Interaction Graph*. This notion is key to our system as it is created, expanded, and leveraged during query execution. To further illustrate this concept, we take the example of enumerating all participants of a conference; the interaction relates in that case to individuals participating to the particular conference; the interaction graph derived from this is a directed graph where the nodes correspond to attendees, and the edges are memories relating attendee A to attendee B. In a sense, *interaction graphs* are overlays sitting on top of *social graphs*.

4.2 Graph Creation

For queries relating to an interaction not yet observed in the system, TransactiveDB tries to discover the implicit interactions connecting the nodes by means of *connection elicitation* and *query routing*. Practically, this requires recursively asking current nodes about further potential connections, and selectively routing the query to those nodes (old or new) that are the most susceptible of contributing new connections.

4.3 Graph Seed Selection

If the query requester did not take part in the interaction, the system engages in a seed discovery to bootstrap the interaction graph by iteratively exploring the social graph (e.g., similarly to the well-known Milgram experiment [5]). The seed is therefore defined as the first person whom the system identifies and who has taken part in the requested interaction. The quality of the seed plays an essential role in the efficiency of the subsequent transactional queries. For instance, selecting a potential "hub", i.e., a person with many outgoing edges to further nodes in the interaction graph, is preferable to selecting more isolated nodes (in that sense, selecting the right seed participant for our conference attendance example might require ranking the potential seeds w.r.t. their connections in that field, their age, or their level of commitment to the conference.)

4.4 Leveraging the Interaction Graph

Interaction graphs are often created as new transactive queries arrive. As time goes by, however, queries relating to the same or similar interaction patterns may surface. TransactiveDB tries to reuse previously elicited interaction graphs for such queries. Considering our previous example, imagine a follow-up query asking for the "list of attendees of the benchmarking workshop that was co-located with the main conference". In this case, the persons holding the information are already listed in the system thanks to the previous transactive query. This new query can hence leverage (at least part of) the interaction graph built for the preceding query.

5. TRANSACTIVE OPERATORS

TransactiveDB borrows from the standard relational algebra for basic operations. It uses operators defined by CrowdDB [3] for crowd-enabled queries. It also supports two new basic transactive operators, namely *TUnion* and *TFill*, and two derivate, *TJoin*, and *TProjection*, in order to handle transactive queries. It is up to the end-user providing the query to specify whether to use the crowdsourcing operations or the transactive memory ones in a declarative fashion. The last parameter of each transactive memory operator is the interaction graph to use as a starting point of its execution; if the input interaction graph is empty, TransactiveDB exploits the seed discovery techniques previously described in Section 4 to bootstrap it. In every case, the result is a pair (R, G) where R is a relation and G is the graph used to obtain the requested data from the transactive memory. We leave as future research deciding how to select what data (columns) to show to the peers in order to obtain the needed information, also according to the privacy settings of the user issuing the query.

All operators are implemented as User Defined Functions (UDFs). Since the system may iteratively interact with its users in order to get missing information, a key focus we take into account is human-readable information. In that sense, tables, attributes and queries should all carry enough textual information to be self-explanatory.

TUnion. The TUnion operator ($T_\cup$) takes a relation R and a starting interaction graph G as input and returns a new relation R' containing all the tuples in R as well as new tuples retrieved transactively from other nodes. The experiment described in [1] [1] is an application of the $T_\cup$ operator and can be formalized by the following relational algebra operation:

$$(all\_iswc\_participants, G') \leftarrow T_\cup(iswc\_participants, G),$$

where *iswc_participants* is the initial relation containing a set of participants to the ISWC conference provided by the user who started the transactive memory experiment (notice that *iswc_participants* may be an empty relation with a specified schema), and G is the graph composed of the people nominated by the user who issued the transactive query.

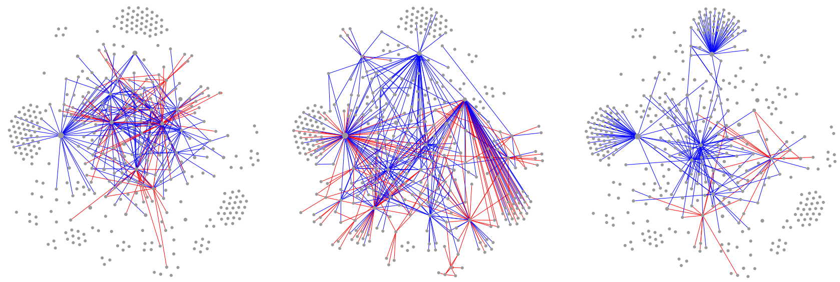
TFill. The TFill operator (T_f) takes as input a relation R, a set of attributes A of R (each featuring a human readable description), and a starting interaction graph G, and exploits the transactive memory features of TransactiveDB to fill all tuples of R with missing values in one or more of A's elements. For example, with the following operations we compute all the participants of the conference that also attended the gala dinner:

$$iswc\_w\_gala \leftarrow all\_iswc\_participants \times \{?\}_{gala}$$
$$(dinner, G'') \leftarrow T_f(iswc\_w\_gala, \{gala\}, G')$$
$$dinner\_only \leftarrow \sigma_{gala=true}(dinner).$$

With the first operation we extend the relation computed previously with an additional column, *gala*, containing only unknown boolean values ("?"). The second operation exploits T_f to obtain the missing values for the new column. Notice that we reuse the graph G' from the antecedent TUnion operation as the starting interaction graph. Finally, in the third operation we select only the people who participated to the gala dinner. This example suggests that the TFill operator can be used in order to make transactive selections of tuples.

TJoin. TFill can be used to define a join operator, TJoin ($T_\bowtie$), which takes as arguments two relations, R and S, a

[1] Reconstructing transactively the list of ISWC attendees.



(a) Iterations 1 and 2 **(b)** Iterations 3 and 4 **(c)** Iterations 5, 6 and 7

Figure 2: Interaction graphs while executing two transactive queries in parallel. Colors encode the new connections elicited by the queries, while the visualization at different iterations shows how the contributions gradually move from central to peripheral peers.

predicate p, and an interaction graph G. This operator can be rewritten in different ways depending on the query and the instance data, opening the door to various query optimization strategies; when R and S are sufficiently small or when the interaction graph is sufficiently large, one can rewrite $T_{\bowtie}$ as T_f on $R \times S$ with p encoded as a new column one wants to fill. In other cases, projecting on the join attributes and running TFill queries to determine the values of those attributes separately before running the join might be more efficient.

TProjection. The TProjection operator, denoted by T_π, is a binary operator that takes as input a relation R and a set of attribute $A = \{a_1, \ldots, a_n\}$. The output produced by $T_\pi(R, A)$ is the projection of the TUnion of R onto the attributes in A, that is,

$$T_\pi(R, A) = \pi_{a_1,\ldots,a_n}(T_\cup(R)).$$

We note that the order of the selection and the TUnion is important: If the system first computes the selection, some contextual information required by the nodes to correctly process the transactive part of the query can be lost. For instance, with $T_\pi(initial\_set, \{email\})$, where $initial\_set = \{(name_1, sname_1, email_1), \ldots, (name_n,\ sname_n, email_n)\}$, the system returns a list of emails from the participants of a conference, while with $T_\cup(\pi_{email}(initial\_set))$ we obtain a generic set of e-mail addresses, since the peers might not have enough context to decide whose e-mails to contribute in that case.

6. RESEARCH AGENDA

A number of fundamental research challenges relating to computer science and social sciences arise in the context of TransactiveDB.

Data Management

- Extending traditional DBMSs architectures (including graph databases) to support the **storage of transactive memories** and interaction graphs. This includes new ways to model transactive data and to provide appropriate indexing mechanisms. For example, it might be necessary to provide some sort of ***forget*** functionality by understanding which memories and interaction graphs are less likely to be reused again.
- Models for **Representing Context** need to be defined to provide users and query processing modules with the most appropriate ways to use their data, e.g., to contextualize information in order for the peers to understand a new request.

Human Computation

- **Source selection** techniques to determine whether the answer could be found in the locally stored data, on the Web, from the crowd, or from a transactive search operation, should be devised.
- Appropriate **seed selection** approaches have to be leveraged to select the first persons in a community to approach to answer a TransactiveDB query. This is necessary to optimize query execution and minimize the number of iterations.

Interdisciplinary Research Areas

- **Incentive mechanisms** have to be created to better exploit the relationships between the peers and the requester; differently from crowdsourcing, the targeted users are known and have some connection to the requester rather than being anonymous. For example, one may provide information because he/she knows this could help a friend or, in an enterprise context, one might want to provide information because a manager is asking for it.
- As human memories are fading-out over time, novel techniques to support human memory with appropriate **Memory cues** have to be designed in order to help people answer queries with the highest recall. An example is to point out different events someone might have attended (e.g., the conference dinner, the paper sessions) in order to help him/her remember who he/she met at a conference.

7. REFERENCES

[1] M. Catasta, A. Tonon, D. E. Difallah, G. Demartini, K. Aberer, and P. Cudré-Mauroux. Hippocampus: Answering Memory Queries using Transactive Search. 23rd International Conference on World Wide Web (WWW 2014), Web Science Track, 2014.

[2] D. E. Difallah, G. Demartini, and P. Cudré-Mauroux. Pick-a-crowd: Tell Me What You Like, and I'll Tell You What to Do. WWW '13, pages 367–374, 2013.

[3] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. CrowdDB: Answering queries with crowdsourcing. SIGMOD '11, pages 61–72, 2011.

[4] A. Halevy, Z. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov. The piazza peer data management system. *Knowledge and Data Engineering, IEEE Transactions on*, 16(7):787–798, July 2004.

[5] S. Milgram. The small world problem. *Psychology today*, 2(1):60–67, 1967.

[6] D. M. Wegner. Transactive memory: A contemporary analysis of the group mind. In *Theories of group behavior*, pages 185–208. 1987.