CareDB: A Context and Preference-Aware Location-Based Database System

Justin J. Levandoski
Supervised by: Mohamed F. Mokbel
Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN
{justin,mokbel}@cs.umn.edu

Abstract— **In this paper, we aim to realize a context and preference-aware database system, *CareDB*, that provides scalable *personalized* location-based services to users based on their preferences and current surrounding context. Unlike existing location-based database systems that answer queries based solely on proximity in distance, *CareDB* considers user preferences and various types of context in determining the answer to location-based queries. To this end, *CareDB* does not aim to define new location-based queries, instead, it aims to redefine the answer of existing location-based queries. The PhD thesis topics covered in this paper solve novel, core systems issues that help realize *CareDB*. These issues are: (1) efficient and extensible core DBMS query processor support for numerous preference evaluation methods, (2) core dbms support for preference query processing in the face of expensive contextual data, and (3) support for *continuous* preference and context-aware query processing.**

I. Introduction

Location-based services aim to provide new services to their users based on the knowledge of their locations. Examples of these services include live traffic reports (*"Let me know if there is congestion within five minutes of my route"*), emergency response (*"Dispatch the nearest five police cars to the crime seen"*), and store finders (*"Where is my nearest restaurant"*). A recent report from ABI Research indicated that the number of location-based services subscribers will be 315 Million by 2011 [1]. The flood of information generated by location-detection devices, along with the large number of mobile users of location-based services, calls for the integration of location-based service functionality with database systems.

Unfortunately, the system semantics of location-based databases are *rigid* as concepts of user "preference" and "context" are ignored. For example, when a user looks for a restaurant, she actually wants to find the "best" restaurant according to her current preferences and context. Existing location-based query processors reduce the meaning of "best" to be the "closest" in terms of pre-computed distances. If desired, preferences and/or context parameters are applied as afterthought queries over the returned result from location-based queries. The *rigidness* of current location-based query processors can be shown with a simple example where a user asks for five restaurants. After retrieving the answer (the nearest five restaurants), the user discovers that the first restaurant has an undesirably long wait, while the second restaurant does not match the user's dietary restrictions. The third restaurant is outside of the user's budget, while the fourth restaurant is closed. Finally, the route to the fifth restaurant is infeasible due to a traffic accident. Location-based services should be *useful*, and a more *useful* set of answers could have been given in the previous example had the database considered user preferences (e.g., dietary restrictions, budget) and contexts (e.g., time of day, traffic, waiting times).

This work is supported in part by the National Science Foundation under Grants IIS0811998, IIS0811935, CNS0708604, and by a Microsoft Research Gift

The goal of my PhD thesis to enable the practical realization of location-based services that embed forms of preference and context in the core query processor of the database sysetem. Thus, *we aim not to define new location-based queries, instead, we aim to redefine the answer of existing traditional location-based queries* by incorporating various types of preferences and context. Due to resource limitations on mobile devices (e.g., small screen), and the fact that users may be in unstable situations (e.g., driving), it is of essence to enhance the quality of the answer and limit the answer to only *useful* tuples according to user preferences and context.

Toward the goal of embedding support for preference and context at the *core* of DBMS, my thesis proposes the *CareDB* system: a context and preference-aware location-based database system. Specifically, my thesis studies three novel *core* systems challenges in realizing *CareDB*:

1) Supporting various preference evaluation methods (e.g., skyline, top-k, k-dominance) at all levels of the query processor in a generic extensible manner, including *core* query operators (e.g., selection and join).
2) Integrating surrounding contextual data (e.g., current traffic, weather) in core preference query processing. Contextual data calls for retrieving some attributes from computationally-intense sources (e.g., third-parties), requiring efficient preference evaluation methods aware of this new cost model.
3) Supporting the distinguishing characteristics of location-based services that include *continuous* queries and dynamic query optimizations in our generic and extensible preference and context-aware query processing engine.

CareDB is a complete database system, meaning all ideas in this thesis have been realized and experimentally evaluated in the PostgreSQL open-source database system. The rest of this paper further describes *CareDB* and its novel contributions.

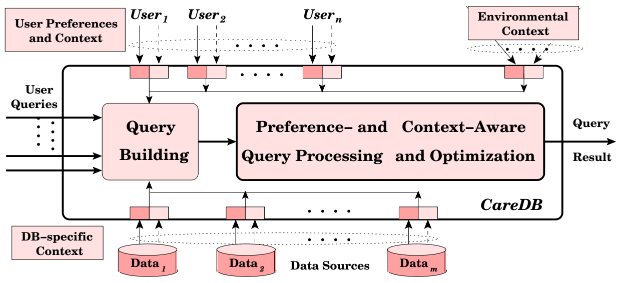


Fig. 1. *CareDB* Architecture

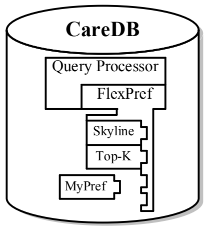


Fig. 2. FlexPref architecture

II. CareDB System Overview

Figure 1 gives an overview of the *CareDB* architecture.

Input. Besides queries, *CareDB* takes preference and contextual data as input. Preferences are specified by a user and stored in a profile. For example, whenever a user searches for a restaurant, her profile may store preferences for travel time, price, rating, and dietary restrictions. *CareDB* is *extensible*, therefore the stored preference semantics are determined by the available preference functions implemented in the query processor (covered in Section III). *CareDB* has three input context types: *user context*, *database-specific context*, and *environmental context*. Each context can be either *static* (rarely changed) or *dynamic* (frequently changing). Static/dynamic context is depicted by solid/dotted lines and dark/light gray rectangles, respectively, in Figure 1. *User context* is any extra information about a user. Static user context data can include income, profession, and age while dynamic attributes include current user location or status (e.g., “at home”, “in meeting”). *Database context* refers to data sources (e.g., restaurant, hotel, and taxi databases) that are registered with *CareDB*. As an example, for a restaurant database, static context data includes price, rating, and operating hours while dynamic context includes current waiting time. *Environmental context* is any information about surrounding environment, assumed to be stored at a third party and consulted by the query processor. A dynamic environmental context includes road traffic, while a relatively static context includes weather information.

Query processing. Upon receiving a query, *CareDB* first injects the query with the stored user preferences. The query is also injected with any contextual data specific to a user (e.g., status) or relevant to the query (e.g., traffic if user is driving).

The preference and context-aware query processing and optimization engine is responsible for executing the query. The main novelty of *CareDB*, and this thesis, lies in this query processing engine. The main responsibilities of this module are to: (1) Embed various types of preference and context-awareness into the existing core of a database system, coupling preference evaluation with traditional database operators (Section III). (2) Support the integration of context-aware query processing, potentially involving expensive calls to third-party contextual data sources (Section IV). (3) Support continuous query processing efficiently by making use of *shared execution* and *incremental evaluation* (Section V).

III. Extensible Support for Preference Methods in the *CareDB* Query Processor

CareDB is *extensible* to a number of existing, and future, preference evaluation methods (e.g., skyline, top-k, k-dominance). Thus, a primary challenge involves pushing the semantics of *each* of these methods *inside* the DBMS query processor in order to realize efficient preference query processing. One approach is to create a user-defined-function that evaluates preference *on-top* of a query plan. A second approach is to create a *custom* implementation for each preference method that can be integrated with query operators. *CareDB* takes a third approach by implementing *FlexPref* [2], a *general* and *extensible* framework for implementing preference evaluation methods inside the query processor. Figure 2 relates the main idea of *FlexPref*. The framework is built into the PostgreSQL query processor. *Only FlexPref* touches the query processor. Each new preference method added to the system is “plugged into” our‘ framework by registering only *four* general functions (defined outside the query processor). The main idea behind this framework is *separation of duties*. (1) The *registered functions*, specific to each preference method, define the *essence* of the preference criteria. These functions define *how* one object is qualitatively better than the other. These functions are *not* aware of the details of the query processor. (2) The *generalized* framework is responsible for efficient preference query processing by injecting preference evaluation as close to the native data operators as possible (i.e., scans, joins). With *FlexPref*, a preference evaluation method can “live” inside the query processor with minimal implementation effort compared to a *custom* approach.

CareDB does not assume data exists in a single table or specialized index in order to execute preference evaluation methods, meaning it must provide efficient execution for arbitrary preference queries involving traditional database operations (e.g., joins). Thus, *FlexPref* is injected into the following operators. (1) Selection. The FlexPref selection algorithm evaluates the set of preferred objects from a *single table*. (2) Join. The FlexPref join algorithm enables efficient preference evaluation for data that exists in *multiple* tables. The main idea behind this join operation involves using the general functions to *prune* tuples from the join input that are guaranteed not to be in the final answer, thus limiting the amount of data that needs to be joined. (3) Index access. FlexPref exploits sorted data from indexes (e.g., B+ tree) by processing sorted attributes in round-robin fashion, and stopping I/O once a stopping condition, provided by the registered general functions, has been met. The main idea is that complete preference answer generation can be guaranteed after reading only a *portion* of the sorted data, thus reducing the I/O overhead compared to query processing over unsorted or non-indexed data.

IV. Query Processing with Expensive Contextual Data in *CareDB*

Context data is any extra data that can help refine preference query answers. For example, traffic data can help predict travel time to a restaurant for a location-based restaurant/store finder. *CareDB* assumes most context data (1) is computed by a *third-party* entity, (2) involves *extensive computation* and communication relative to processing local data, and (3) is computed in a *state-less* manner, i.e., assumes any third-party entity will not have knowledge of previous *CareDB* requests. The *CareDB* query processor is designed to take these challenges into account by providing efficient preference query processing over contextual data. Furthermore, this query processing framework is *general*, capable of handling a number of preference methods (e.g., skyline, multi-objective).

The goal of preference query processing over contextual data is to minimize *unnecessary requests* for expensive contextual attributes, i.e., requests for expensive attributes not belonging to records in the final preference answer. A naive approach requests all necessary data from a remote source, and computes the preference answer using existing preference algorithms that assume all *local* data. This approach is unnecessarily expensive, as it makes the maximum amount of unnecessary requests.

CareDB takes a more efficient approach for preference query processing over expensive contextual data, consisting of three phases. Phase I This phase takes as input the dataset D, and forms an initial query answer (abbr. S_c) by running the preference query over the local (i.e., "cheap") attributes, while also ensuring that objects in S_c are guaranteed to be in the final query answer. Then, it performs a random access request to retrieve the "expensive" attributes for objects in S_c. Phase I does not incur unnecessary requests as it retrieves only the expensive attributes for those objects that are guaranteed to be in the final answer. Phase II: This phase takes as input the dataset D from Phase I and performs three main operations: (a) Making a range request to retrieve the expensive attributes for a small sample of objects that are *not* in the initial answer S_c, (b) Creating a pruning set P by combining the returned objects from the range request with some of the objects in S_c. A set of objects $M \subseteq (P - S_c)$ are added to the final preference answer at this point. (c) Using P to prune a set of incomplete objects L (i.e., objects without their expensive attributes) that are guaranteed not to be in the final answer regardless of their expensive attribute values. Thus, the efficiency of this framework depends on *maximizing* the number of objects in L, as the query processor can skip requests (i.e., expensive computation and communication) for objects in L. Phase III: This is a final *cleaning* phase takes as input the dataset $(D - L)$, and computes a final answer by first making a random request for objects in $D - (L \cup S_c)$ without their contextual (i.e., expensive) attributes. These remaining objects are then *cleaned*, a process that discards dominated objects. Any *non-cleaned* objects are added to the final preference answer S_c. Ideally, Phase III is unnecessary as all incomplete (and non-preferred) objects would be pruned by Phase II.

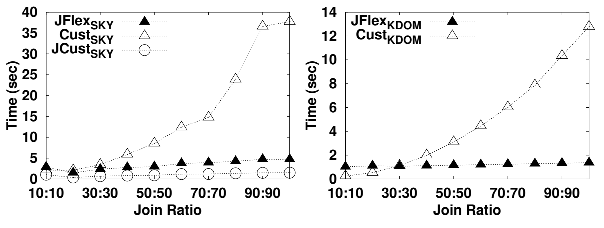
V. Continuous Queries in *CareDB*

Many traditional database applications rely on *snapshot* queries that are terminated once the answer is returned to the user. Conversely, location-based applications make use of both snapshot and *continuous* queries, that stay active at the server side until explicitly terminated. Traditional location-based continuous queries react to two types of dynamic data: (1) Moving objects, where movement of the queried data causes changes to the answer, and (2) Moving queries, where movement of the *user query*, and not necessarily the queried data, causes changes to the answer. In addition to moving objects and moving queries, *CareDB* handles a third query paradigm: changes in contextual and preference data. In this paradigm, changes in context or preference data may cause a query answer to change, even if object and query locations remain stable.

Due to long-running continuous queries, location-based databases employ two main query processing paradigms. (1) *Incremental evaluation.* The incremental evaluation paradigm only evaluates changes to the query *answer*, rather than repetitively re-evaluating a query. In current systems, incremental evaluation takes place based on changes to the *location* of a query or object. (2) *Shared execution.* The shared execution paradigm shares common data between queries, in location-based databases, this task is abstracted by a join between a set of queries and data objects. Currently, only object *location* is shared between queries in location-based databases. *CareDB* adds the notion of context and preference to the incremental evaluation and shared execution paradigms.

VI. Related Work

Following several theoretical works for expressing user preferences in database systems [3], [4], [5], recent systems have been developed to include preference and context in databases. Examples of these systems include PREFER [6], PreferenceSQL [7], Personalized queries [8], and contextual database [9], [10], [11]. The PREFER system [6], [12] incorporates preferences into a single weighted ranking function where preferred results are generated by finding pre-computed materialized views whose weight function is similar to the query. PreferenceSQL [7], [13] provides new constructs for expressing preference in SQL, rules for combining preferences in a *cascading* or *pareto-accumulation* manner, and rules for translating PreferenceSQL into traditional SQL queries. Personalized queries [14], [8], [15] model preferences using a degree of interest score, where queries are injected with mandatory and secondary preferences based on this score. The resulting query is built using traditional SQL constructs. Contextual database [9], [10], [11] focuses on modeling



(a) Skyline (b) K-Dominance

Fig. 3. Join experiment: FlexPref vs. a specialized approach

contextual preferences, and integrating context into query definitions. *CareDB* distinguishes itself from all these systems as it: (a) provides a full-fledged realization of preference and context-aware databases, (b) goes beyond preference modeling and query rewriting to address processing preferences and context at the query operator levels, (c) exploits a built-in approach where the preference and context-aware processing are embedded inside core processing of query operators, unlike other systems that build personalization and context-management modules *on-top* of existing relational databases, and (d) *CareDB* is equipped with the necessary modules that support the special characteristics of location-based servers, e.g., continuous queries and dynamic environments.

VII. Initial Experimental Performance

To demonstration the feasibility and advantages of *CareDB*, this section provides a preliminary experiment that shows the benefit of *FlexPref* (Section III). The experiment implements both the skyline and k-dominance preference method within *FlexPref*, denoted Flex$_{SKY}$, Flex$_{KDOM}$, respectively. We also implemented the skyline operator (Cust$_{SKY}$) [16], the two-scan K-dominance algorithm (Cust$_{KDOM}$) [17], and the the custom skyline join operator (JCust$_{SKY}$) [18] in order to fairly evaluate the *FlexPref*'s multi-relational preference execution framework. *FlexPref*, along with JCust$_{SKY}$ [18], are implemented in the backend executor of the PostgreSQL 8.3.5 open-source database [19]. All other algorithms are implemented as extensible user-defined functions in PostgreSQL, the fairest implementation method for our counterparts as they are designed to execute *on-top* of a query plan.

Figures 3(a) and 3(b) give the runtimes for skyline and k-dominance methods, respectively, for a binary join as the join ratio increases, i.e., the cardinality of both input tables (with 1K distinct keys) increases from 10K to 100K. These results clearly highlight the advantages of *FlexPref*. The optimized FlexPref implementations exhibit scalable behavior as the join ratio (and data size) increases. FlexPref is superior to the Cust$_{SKY}$ and Cust$_{KDOM}$ methods that represent an *on-top* approach for the multi-table case. Both Cust$_{SKY}$ and Cust$_{KDOM}$ cannot reduce the input to the join, thus must process the complete join result. Interestingly, JFlex$_{SKY}$ exhibits comparable performance to the *custom* skyline join JCust$_{SKY}$. These results are promising, and show that (1) FlexPref is clearly advantages for arbitrary DBMS queries compared to an *outside* (or *on-top*) approach and (2) competitive with *specialized* approaches for more sophisticated queries.

VIII. Conclusion

This paper outlined a PhD thesis that proposes *CareDB*, a full-fledged location-based DBMS capable of *scalable preference and context-aware query processing*. The basic architecture was first presented that provided the *input* (user preferences and differing contexts) and query processing objectives of *CareDB*. Three core systems challenges, necessary to realizing *CareDB*, along with initial approaches to solving these challenges were presented. These challenges include (1) supporting various preference methods *inside* the query processor in a generic extensible manner, (2) providing the query processor with a general framework to efficiently process preference queries in the face of (some) contextual data attributes that require extensive computational and communication costs, and (3) supporting *continuous* preference and context-aware queries in *CareDB*. The novelty of *CareDB* was demonstrated by classifying the state-of-the-art related work in preference and context query processing. Finally, initial experimental evidence was given demonstrating the feasibility and advantages of *CareDB*'s built-in approach to preference and context-aware query processing.

References

[1] "ABI Research. GPS-Enabled Location-Based Services (LBS) Subscribers Will Total 315 Million in Five Years. http://www.abiresearch.com/abiprdisplay.jsp?pressid=731. September, 27, 2006."
[2] J. J. Levandoski, M. F. Mokbel, and M. E. Khalefa, "FlexPref: A Framework for Extensible Preference Evaluation in Database Systems," in *ICDE*, 2010.
[3] R. Agrawal and E. L. Wimmers, "A Framework for Expressing and Combining Preferences," in *SIGMOD*, 2000.
[4] J. Chomicki, "Preference Formulas in Relational Queries," *TODS*, vol. 28, no. 4, pp. 427–466, 2003.
[5] M. Lacroix and P. Lavency, "Preferences: Putting More Knowledge into Queries," in *VLDB*, 1987.
[6] V. Hristidis, N. Koudas, and Y. Papakonstantinou, "PREFER: A System for the Efficient Execution of Multi-parametric Ranked Queries," in *SIGMOD*, 2001.
[7] W. Kießling, "Foundations of Preferences in Database Systems," in *VLDB*, 2002.
[8] G. Koutrika and Y. Ioannidis, "Personalization of Queries in Database Systems," in *ICDE*, 2004.
[9] K. Stefanidis and E. Pitoura, "Fast Contextual Preference Scoring of Database Tuples," in *EDBT*, 2008.
[10] K. Stefanidis, E. Pitoura, and P. Vassiliadis, "A Context-Aware Preference Database System," *International Journal of Pervasive Computing and Communications*, vol. 3, no. 4, pp. 439–460, 2007.
[11] ——, "Adding Context to Preferences," in *ICDE*, 2007.
[12] V. Hristidis and Y. Papakonstantinou, "Algorithms and Applications for Answering Ranked Queries using Ranked Views," *VLDB Journal*, vol. 13, no. 1, pp. 49–70, 2004.
[13] W. Kießling and G. Köstler, "Preference SQL: Design, Implementation, Experiences," in *VLDB*, 2002.
[14] G. Koutrika and Y. Ioannidis, "Constrained Optimalities in Query Personalization," in *SIGMOD*, 2005.
[15] G. Koutrika and Y. E. Ioannidis, "Personalized Queries under a Generalized Preference Model," in *ICDE*, 2005.
[16] S. Börzsönyi, D. Kossmann, and K. Stocker, "The Skyline Operator," in *ICDE*, 2001.
[17] C.-Y. Chan, H. Jagadish, K.-L. Tan, A. K. Tung, and Z. Zhang, "Finding k-Dominant Skylines in High Dimensional Space," in *SIGMOD*, 2006.
[18] W. Jin, M. Ester, Z. Hu, and J. Han, "The Multi-Relational Skyline Operator," in *ICDE*, 2007.
[19] "PostgreSQL: http://www.postgresql.org."