

Periscope/SQ: Interactive Exploration of Biological Sequence Databases

Sandeep Tata

Willis Lang
University of Michigan
Ann Arbor, Michigan, USA
{tatas, wlang, jignesh}@umich.edu

Jignesh M. Patel

ABSTRACT

Life science laboratories today have to rely on procedural techniques to store and manage large sequence datasets. Procedural techniques are cumbersome to use and are often very inefficient compared to optimized declarative techniques. We have designed and implemented a system called Periscope/SQ that makes it possible to rapidly express complex queries within a declarative framework and take advantage of database-style query optimization. As a result, queries in Periscope/SQ run orders of magnitude faster than typical procedural implementations. We demonstrate the power of Periscope/SQ through an application called GeneLocator which allows biologists to rapidly explore large genomic sequence databases.

1. INTRODUCTION

The life-sciences community today produces increasingly large amounts of sequence data thanks to tremendous advances in high throughput technologies. Genomes of several organisms have been sequenced, and several more sequencing projects are underway. GenBank [4], the genomic sequence repository operated by the NCBI, doubles in size every 16 months! However, laboratories around the world still employ custom procedural methods using JAVA, Perl, or Python code to manage this data and process complex queries. The drawbacks of a procedural querying paradigm are twofold: a) severe limitations on the ability of a scientist to rapidly express sophisticated queries, and b) the lack of an infrastructure for optimization which often results in inefficient plans with very slow query response times. With increasing data sizes, there is an urgent need for a database approach that allows scientists to focus on the biology by taking away the burden of query processing. In [9], we outlined the design of Periscope/SQ, a system that solves this problem by enabling declarative and efficient querying on biological sequence datasets.

Biologists analyze DNA and protein sequence databases in several complex ways. With DNA sequences (a string

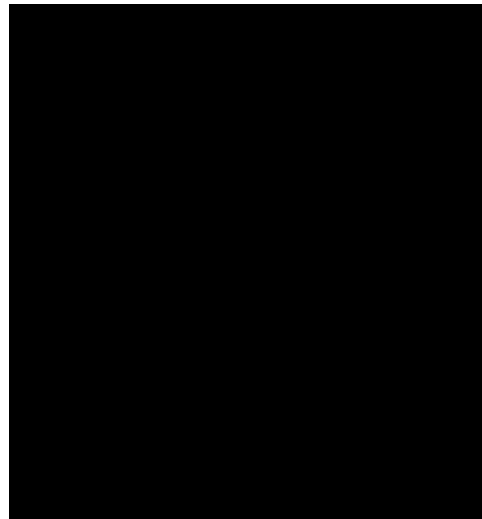


Figure 1: Architecture of Periscope/SQ.

over $\{A, C, G, T\}$), scientists perform approximate similarity searches to identify regions of special interest such as genes, regulatory markers, repeating units, etc. In the case of protein databases, scientists are often interested in locating proteins that are similar to a target protein of interest. This similarity may include the primary sequence (over 20 basic amino acids), or the local folding patterns in the secondary sequence (alphabet of size three: α -helix, β -sheet, or loop), or a combination of the two. The criteria for specifying similarity are often approximate and the desired output is usually an ordered list of results.

In this presentation, we describe the architecture of the Periscope/SQ system and demonstrate a sequence querying application called GeneLocator. GeneLocator allows biologists to express complex queries with both sequence matching predicates and traditional relational predicates using a web-based interface. By executing these queries orders of magnitude faster [9] than current procedural algorithms, Periscope/SQ enables scientists to interactively pose queries and examine the results. Consequently, Periscope/SQ enables a paradigm shift where scientists can now think in terms of exploring the model space instead of being limited to exploring the data space.

This proposal is organized as follows: In Section 2, we describe the architecture of Periscope/SQ. In Section 3, we describe the details of the demonstration. We summarize our demonstration proposal and conclude in Section 4.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permission from the publisher, ACM.

VLDB '07, September 23-28, 2007, Vienna, Austria.

Copyright 2007 VLDB Endowment, ACM 978-1-59593-649-3/07/09.

2. SYSTEM ARCHITECTURE

Queries posed by life scientists often involve complex sequence similarity conditions in addition to regular relational operations like select, project, join, and aggregate. As a result, we chose to implement Periscope/SQ as an extension of an existing object-relational DBMS [7] instead of building a sequence-only DBMS from the ground up. We built Periscope/SQ by extending the free open source ORDBMS Postgres [6].

Periscope/SQ is based on an extension of relational algebra called the Protein Query Algebra (PiQA [8]). The query language is called Protein Query Language (PiQL [9]), and is an extension of SQL. In this demonstration, we focus on querying DNA sequences (though the full versions of these languages also permit querying on protein sequences).

The overall architecture of Periscope/SQ is shown in Figure 1. The application issues queries in PiQL which are parsed by the PiQL parser. The Periscope/SQ optimizer rewrites this query using the algebraic properties of PiQA and cost estimates for different plans. The query is then passed on to Postgres for relational optimization and execution. The Postgres engine takes advantage of several Periscope/SQ Abstract Data Types (ADTs) and User-Defined Functions (UDFs) to execute the query plan.

2.1 Query Language: PiQL

The extensions to PiQA described in [8] include a new datatype called match. There are several algebraic operators that produce and consume these match types. This is of course, in addition to the basic relational data types such as integers, string, and the tuple type. We describe the key new data types, operators and their implementation in Periscope/SQ below.

2.1.1 Hit and Match Types

Hit: A hit is basically a triple (p, l, s) . When specified together with some sequence, the hit (p, l, s) means that there is a *hit* at position p of length l with a score of s on the given sequence. For instance, suppose that $A = (2, 3, 3)$ is a hit on the sequence $SEQ = \text{GGTTTAGGAGGTA}$. This hit refers to the **GGT** substring, which could have matched some query with a score of 3. This hit can be shown in the original database sequence as **GGTTTAGGAGGTA** with the hit highlighted in bold-face.

Match: A match is simply a set of hits. For example, consider the sequence $SEQ = \text{GGTTTAGGAGGTA}$ and a query to **GGT** followed by a **GGA** within 10 symbols. A match for this query using an exact matching paradigm is $X = \{(2, 3, 3), (8, 3, 2)\}$. This match describes two hits in the data sequence as shown in bold-face in **GGTTTAGGAGGTA**.

The Match type is implemented as an array of integers. Several functions are defined on the match type. Key functions include *match_augment* (to combine two matches that are within a specified distance) and *match_contains* (to compute matches that contain another match). Readers interested in details of other functions are referred to [8, 9].

2.1.2 Match Operator

The Match operator finds approximate matches to a query string. Since local similarity search is a crucial operation in querying biological sequences, one needs to pay close at-

Function	Description
Nest	Nests matches in different rows with a common attribute into a single match
Unnest	The reverse of Nest
Augment	Combines matches from two tables if they have the same id into one match if they are within the specified distance
Contains	Similar to augment, except it checks to see if one match is contained in the other

Table 1: PiQL Functions

tention to the match model. In practice, the commonly used match models include exact match model, k -mismatch model, and the substitution matrix based models. A k -mismatch model allows for at most k substitutions (mismatches) between the query and the match sequence. The exact match model is merely the k -mismatch model with $k=0$. The general substitution matrix based models use a matrix that specifies the precise score to be awarded when one symbol in the query is matched with a different symbol in the database. For a more detailed discussion of various matching models, we refer the reader to [3].

The match operator is implemented as a User-Defined Table-Function in PostgreSQL that takes as inputs: *tblname*, *idcol*, *seqcol*, *model*, *query*, and *cuto*. Here, *tblname* is the name of the table being searched, *idcol* is the name of a key attribute in this table, *seqcol* is the name of the sequence attribute, *model* is the name of the match model (exact, k -mismatch, or matrix), *query* is the query string or a pointer to the query matrix, and finally, *cuto* is the threshold score below which matches are not included in the result.

Several other functions are available in PiQL. A brief description of the key PiQL functions is provided in Table 1. For instance, *Augment* and *Contains* provide a concise way of expressing operations on an entire table of matches. Consider the following query:

```
SELECT m1.seqid, m1.matchid,
match_augment(m1.match, m2.match, 0, 100) FROM
MATCH(T1, seqid, sequence, 0, 'GAC') m1,
MATCH(T1, seqid, sequence, 1, 'TACAGGG') m2,
WHERE m1.seqid = m2.seqid
```

This query can be concisely expressed in PiQL as:

```
SELECT * FROM AUGMENT(
MATCH(T1, seqid, sequence, 0, 'GAC')
MATCH(T1, seqid, sequence, 1, 'TACAGGG') 0, 100)
```

2.2 Optimizer

The Periscope/SQ optimizer chooses an efficient implementation for each match operator in the query, and also considers an optimal ordering for evaluating each match operation. The functions that implement the match operation are summarized in Table 2. The algorithms used in *match_st_km* and *match_st_matrix* are based on a best-first exploration of a search tree and are similar to the algorithms used in [2]. The techniques used by the optimizer to estimate selectivities of different predicates and the cost of different operators are described in [9]. In particular Periscope/SQ employs a linear time algorithm to optimize the plan for a set of sequence predicates separated by specified distance.

Function	Description
match_scan_km	Scan-based algorithm for k -mismatch
match_st_km	Su x tree-based algorithm for k -mismatch
match_st_matrix	Su x tree-based algorithm for matrix searches
match_st_aug_rscan	Match with Su x tree, then augment to the right with a scan
match_st_aug_lscan	Match with Su x tree, then augment to the left with a scan

Table 2: Evaluation Functions for Match

The algorithm compares the cost of choosing the fastest algorithm for each predicate independently and then joining them against the cost of rewriting adjacent predicates with the match and augment operator as described in [9]. In [9], we show that for typical queries with few (3%) predicates, this linear time algorithm produces plans that run in within 6% of the running time of the optimal plan.

2.2.1 Indexes

One of the strength of Periscope/SQ is its ability to leverage su x tree indexes for evaluating sequence predicates. Periscope/SQ includes functions to build a su x tree index on a table by specifying the name of the table and the sequence attribute in the table. The su x tree is built using the TDD algorithm described in [10].

Periscope/SQ keeps track of the su x tree indexes available on different tables and attributes in a catalog (implemented as a relational table).

3. DESCRIPTION OF DEMONSTRATION

In this demonstration, we will illustrate two central features of Periscope/SQ:

1. The ability to express sophisticated sequence queries in the context of other relational operations, and
2. The performance advantage of using a declarative framework that leverages multiple access paths and an optimizer.

We demonstrate these aspects using a sample application called GeneLocator. GeneLocator uses Periscope/SQ to process nucleotide sequence queries that are constructed using a web based front-end (Figure 2). This application is being used to search DNA sequences for occurrences of patterns (or groups of patterns) in the so called promoter regions² upstream of known genes. The basic idea is that given the binding site signature of a transcription factor [1], one can locate likely targets of regulation using sequence similarity search. Additional constraints like restricting the search to targets that are expressed in specific tissues or finding genes that are targets in multiple organisms can also be

Table Name	Schema
org_promoters	(seqid, genename, promoter)
org_genes	(genename, chromosome, start, end)
org_expression	(genename, tissue, exp_lvl, total)

Table 3: Tables in GeneLocator

Figure 2: GeneLocator Query Screen

used. Readers interested in learning more about transcription, regulation, and detecting targets may refer to [1].

3.1 Datasets

The sequence data and the annotation data for GeneLocator was obtained from NCBI [5]. Annotation tables that provided the coordinates of all the genes, and tissue expression data were also downloaded from NCBI. Subsequences of length 5000 symbols were extracted from the upstream region of each annotated gene and inserted into a table of promoters for each organism. The simplified schemas of the tables used in GeneLocator are shown in Table 3.

3.2 Application Interface

GeneLocator has a web based interface that allows a biologist to construct the necessary PiQL queries. Figure 2 shows this interface. A scientist starts with a target organism and checks off other genomes to include additional evidence. She then specifies up to four sequence patterns to look for in the promoter regions of the genes. Patterns can be exact matches, patterns tolerating up to k -mismatches, or matches to a position weight matrix (PWM) specified by an uploaded file. These patterns can be constrained by specifying a distance (range) from the transcription start site or from the previous pattern. When the query is submitted, a PiQL query is constructed and sent to an instance of Periscope/SQ.

Example: Consider an example where a scientist chooses the mouse genome as the target genome, and chooses the human genome for additional search in the GeneLocator interface. Next, she specifies two patterns, a 2-mismatch pattern `ATTACATACATA` at a distance of 0-1000 symbols from the transcription start site, and an exact pattern `CCCGC` at a distance of 40-1000 symbols downstream of the previous pattern. She also checks the tissue expression box for Eye. GeneLocator generates the following PiQL statement in response:

Figure 3: GeneLocator Result Screen

```
SELECT * FROM AUGMENT( MATCH(mouse_promoters, genename,
promoter, ATGATTACATACATA),
MATCH(mouse_promoters, genename, promoter, ATG
ATGCGC, 40, 1000) as ma , mouse _expression mx
WHERE PROMOTER_LENGTH - match_start(ma.match)
BETWEEN 0 and 4000 AND
ma.seqid = mx.genename AND mx.tissue=Eye)
```

The optimizer rewrites this query to:

```
SELECT * FROM match_st_aug_rscan(
mouse_promoters, genename, promoter, ATGATTACATACATA,
ATGATGCGC, 40, 1000) as m1, mouse _expression mx
WHERE PROMOTER_LENGTH - match_start(m1.match)
BETWEEN 0 and 4000
AND m1.seqid = mx.genename AND mx.tissue=Eye)
```

A similar query is produced for the human genome. These queries are executed and the results are displayed as shown in Figure 3. The first and second columns list the target genes from the human and mouse genomes that are expressed in eye tissue. The third column, computed as a intersection between the first two columns, contains results common to both genomes and is of particular interest to biologists. The power of being in a relational setting can easily be leveraged to further process these results to check for genes that are also expressed in other tissues, or other organisms, or to sort these lists in different ways.

By clicking on the name of the gene, the user is taken to an NCBI page about the gene. The distance from the trascription start site is linked to the NCBI mapviewer site that displays the relevant region of the mouse genome (Figure 4). Query times range from seconds to one to two minutes depending on complexity and are one to two orders of magnitude faster than those with procedural scan-based approaches [9]. Since queries often start out simple and are iteratively refined, the response time can be further improved by making use of materialized views and caching the results of previous queries.

4. SUMMARY

In this presentation, we describe the design and implementation of the Periscope/SQ system and show that it enables interactive querying on large biological sequence databases. Interactive querying on large sequence databases enables a

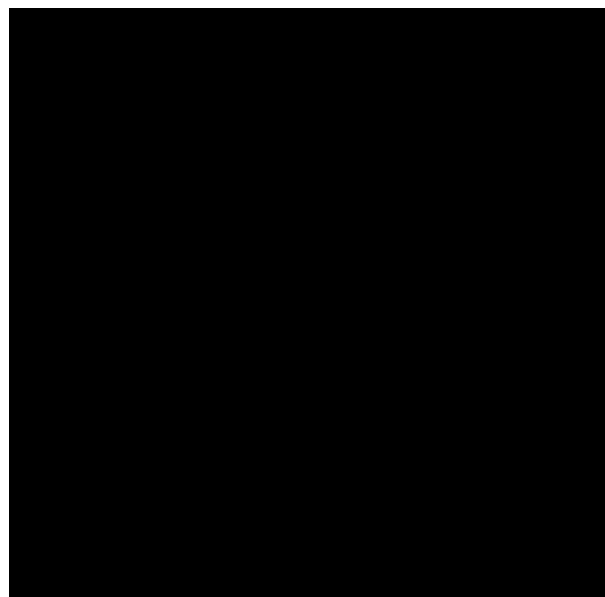


Figure 4: Mapviewer from NCBI

paradigm shift from limiting the scientist to data space exploration to enabling a model space exploration, where they can rapidly pose and refine complex queries that correspond to biological models. We demonstrate an application on top of Periscope/SQ that permits complex queries over DNA sequences and provides fast response times.

5. ACKNOWLEDGEMENTS

This research was primarily supported by the National Science Foundation under grant DBI-0543272 and by an unrestricted research gift from Microsoft.

6. REFERENCES

- [1] B. Alberts, A. Johnson, J. Lewis, M. Ra , K. Roberts, and P. Walter. *Molecular Biology of The Cell*. Garland Science, 4th edition, 2002.
- [2] B. Dorohonceanu and C. Nevill-Manning. Accelerating protein classification using su x trees. In *ISMB*, pages 1281-1283.
- [3] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis : Probabilistic Models of Proteins and Nucleic Acids*. Cambridge Univ. Press, 1st edition, 1999.
- [4] Growth of Genbank. www.ncbi.nlm.nih.gov/Genbank.
- [5] NCBI Genomes. <ftp://ftp.ncbi.nlm.nih.gov/genomes/>.
- [6] PostgreSQL. www.postgresql.org.
- [7] M. Stonebraker, D. Moore, and P. Brown. *Object Relational DBMS: Tracking the Next Great Wave*. Morgan Kaufman, 2nd edition, 1999.
- [8] S. Tata and J. M. Patel. PiQA: An Algebra for Querying Protein Data Sets. In *SSDBM*, pages 141-150, 2003.
- [9] S. Tata, J. M. Patel, J. S. Friedman, and A. Swaroop. Declarative Querying for Biological Sequences. In *ICDE*, pages 87-98, 2006.
- [10] Y. Tian, S. Tata, R. Hankins, and J. Patel. Practical methods for constructing su x trees. *VLDB Journal*, 14:281-299, September 2005.