StarDB: A Large-Scale DBMS for Strings

Majed Sahli
King Abdullah University for Science & Technology
Thuwal, Saudi Arabia
majed.sahli@kaust.edu.sa

Essam Mansour
Qatar Computing Research Institute
Doha, Qatar
emansour@qf.org.qa

Panos Kalnis
King Abdullah University for Science & Technology
Thuwal, Saudi Arabia
panos.kalnis@kaust.edu.sa

ABSTRACT

Strings and applications using them are proliferating in science and business. Currently, strings are stored in file systems and processed using ad-hoc procedural code. Existing techniques are not flexible and cannot efficiently handle complex queries or large datasets. In this paper, we demonstrate StarDB, a distributed database system for analytics on strings. StarDB hides data and system complexities and allows users to focus on analytics. It uses a comprehensive set of parallel string operations and provides a declarative query language to solve complex queries. StarDB automatically tunes itself and runs with over 90% efficiency on supercomputers, public clouds, clusters, and workstations. We test StarDB using real datasets that are 2 orders of magnitude larger than the datasets reported by previous works.

1. INTRODUCTION

Analytics are performed on a single long string, such as the human genome, or a large collection of short strings, such as DNA reads [7]. Due to low storage costs, large collections of strings are accumulating in academic and industrial research labs [2]. Governmental and industrial bodies aim at sequencing individuals with cancer, rare diseases, and infectious diseases. Ambitious projects include the 100,000 Genomes Project[1] in the UK and the Cancer Genome Atlas[2] in the US. Several medical breakthroughs are awaiting the proper management and processing of the growing genetic data banks. Similarly, large collections of sequential signals from the world's largest radio telescope[3] will require extensive processing. Moreover, textual data (e.g., World Wide Web content) and time series (e.g., stock prices) are examples of important sequential data [7].

Managing strings is still done in an ad-hoc fashion using application-specific methods. Complex queries and analytics require using different systems and moving data between them. For example, biologists use BLAST[4] to find regions of local similarity between biological sequences and KAT[5] to analyze substring frequency spectra. Several attempts were proposed to handle strings using a DBMS approach. Periscope/SQ [11] extended PostgreSQL with matching operations over biological sequences. It is challenging to express common string queries, such as motifs and k-mers, with only matching operations. Moreover, complex queries require the efficient utilization of large infrastructures to finish in reasonable times. Therefore, Periscope/SQ reported simple matching queries over sequences of 5,000 symbols.

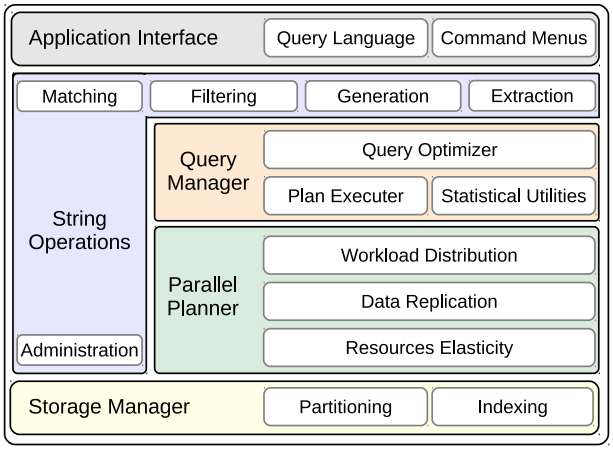


Figure 1: Architecture of StarDB.

In this demonstration, we describe the architecture of StarDB, a large-scale DBMS for strings. StarDB uses our novel data structures [9] and parallel string algorithms [10] to natively facilitate large-scale analytics for strings. We incorporate our automatic tuning framework for large infrastructures [8] to meet users time and budget constraints. StarDB allows users to easily form complex string queries. It hides the complexity of supercomputers, large clusters, public clouds, and multicore machines from users.

The rest of this paper is organized as follows. Section 2 discusses the system architecture and how it supports large infrastructures. In Section 3, we detail our demonstration. Section 4 compares StarDB to procedural systems.

[1]http://www.genomicsengland.co.uk
[2]http://cancergenome.nih.gov
[3]http://www.skatelescope.org

[4]http://blast.ncbi.nlm.nih.gov/
[5]http://www.tgac.ac.uk/KAT/

Proceedings of the VLDB Endowment, Vol. 8, No. 12

2. SYSTEM ARCHITECTURE

StarDB is a distributed system built over the master-worker parallel programming paradigm. Complex queries are solved in parallel whereas simple ones are run in serial. StarDB utilizes a scalable data model and exposes a declarative query language while hiding infrastructure complexity. Figure 1 shows the overall system architecture.

Naïvely, operations require multiple data scans and mostly involve a much wider search space than the strings themselves. Moreover, string operations workloads scale superlinearly with data size [5]. We attack these challenges at different system layers. (*i*) We provide a declarative query language that includes a comprehensive set of optimized string operations. (*ii*) We use efficient and self-tuned parallel algorithms to utilize resources and speedup computation. (*iii*) We employ full-text indexes in the form of tries to strike a balance between index features and sizes; and between preprocessing and information retrieval times. Hence, StarDB natively supports string operations, handles real datasets, and efficiently utilizes various infrastructures.

Example 1: A user needs to find text that appears frequently in Wikipedia. The user has to work around spelling mistake and simple differences such as noun plurals and verb tenses. First, the Wikipedia archive is imported into StarDB. StarDB indexes the dataset and may partition or replicate indexes depending on size and available resources. Motifs are patterns that appear frequently but not necessarily exactly. StarDB supports edit distance or hamming distance for approximate matching. To find all frequent patterns, a query to generate motifs is used. Running on 480 cores, the motifs search space (a combinatorial tree over English alphabet) is partitioned to 17,576 tasks (26^3 subtrees). The workload is balanced by dynamically assigning tasks and the results are gathered and returned to the user.

2.1 Data Model and Query Language

Strings are ordered sets of symbols, grouped into collections. Depending on how collections are generated, they could consist of several long strings or millions of short ones. Our data model, StarDM, is to be efficient and scalable in terms of string length and collection size. Generally, operations in StarDM take one or two collections as input and output a collection. Our query language, StarQL is declarative. The unification of format in StarDM allow for pipelining operations to express complex queries in StarQL.

We introduce five categories for operations in StarDB. (*i*) *Administrative* commands are used to manage data collections (e.g., the `import` StarQL construct). (*ii*) *Matching* operations take 2 collections as input and output a collection of strings that represent the substrings from the left operand that match a string in the right operand. Example StarQL constructs include `exact` and `approximate`. (*iii*) *Filters* are used to operate on parts of collections of certain features, such as `length`. (*iv*) *Generation* operations result in new string collections for common and repeated motifs and k-mers. (*v*) *Extraction* operations are used to create windows over strings given absolute positions (e.g., `range`) or relative ones (e.g., `prefixes`).

In addition, basic set operations (e.g., union) and aggregate operations (e.g., `count`) are supported. StarQL also supports sorting and returning partial results using the `limit` keyword. The syntax of StarQL is SQL-like. Figure 2 depicts the syntax of selection queries.

```
<query>     := <select> | <import> | <delete> | <update>
<extractor> := PREFIXES | SUFFIXES | RANGE
<generator> := RMOTIFS | CMOTIFS | KMERS
<identifier>:= <path> | <name> | <import>
<filter>    := PREFIX | SUFFIX | SUBSTRING | <metadata>
<match>     := EXACT | APPROXIMATE | REGEX
<select>    := SELECT <extractor> | <generator> | *
               FROM <identifier> | (<query>)
               [ WHERE <filter> | <match> | <metadata> ]
               [ <aggregate-ops> | <limit> | <order-by>]
```

Figure 2: Abstract BNF of StarQL selection query.

In the example of finding motifs from Wikipedia, the user may decide to filter out motifs of length 4 or less as they correspond to common short words, such as articles and prepositions. The user allows an edit distance of 2 characters so words like "fishes" and "fishy" count as occurrences for the motif "fish". The length and approximate matching parameters are readily available in StarQL. The user in our example may form and submit the following StarQL query.

```
SELECT RMOTIFS FROM wiki WHERE LEN>4
  AND EDIT<=2 AND FREQ>1000;                                (1)
```

StarDB users can form complex queries easily. Consider two collections of strings, `hom` for human DNA and `mus` for mouse DNA. A biologist finds the sequences that are similar in both collections by using the approximate matching operation. For simplicity, we assume a hamming distance of 3 is acceptable. The following StarQL query results in all the subsequences in `hom` that have approximate matches in `mus`.

```
SELECT * FROM hom WHERE APPROXIMATE(mus)
  AND HAMM<=3;                                              (2)
```

2.2 Optimization and Execution

In StarQL semantics, the final output is always a subset of the operation after a `SELECT` keyword. Conditions after a `WHERE` keyword are interpreted from left to right, but not necessarily executed in the same order. By this convention, it is clear to users how StarQL is interpreted. They form queries that represent their intentions. The costs of valid plans are compared internally to choose the most efficient plan. The following nested query extracts the suffixes of length 3 from the Wikipedia repeated motifs.

```
SELECT SUFFIXES FROM (SELECT RMOTIFS FROM wiki
                      WHERE LEN>4 AND EDIT<3)               (3)
  WHERE LEN=3;
```

Alternatively, the motifs query could be saved as a collection in StarDB for further processing. This allows users to analyze intermediate results and execute different queries without having to start from scratch. For example, after extracting suffixes of length 4 from the motifs, the user may want to explore prefixes or suffixes of different lengths. In this case, first the motifs are saved as a collection in StarDB then used in subsequent queries as shown next. Motifs of length 5 will not contribute to the results of the prefixes.

```
IMPORT ( SELECT RMOTIFS FROM wiki
          WHERE LEN>4 AND EDIT<3 ) AS mot;
SELECT SUFFIXES FROM mot WHERE LEN=10;                      (4)
SELECT PREFIXES FROM mot WHERE LEN>5 AND LEN<10;
```

2.3 Indexing and Large-scale Parallelism

Scaling to large infrastructures, we need flexible data structures, efficient parallel algorithms, and automatic tuning. Given a query plan and a user budget, StarDB decides an execution plan. While serial execution is preferred in the case of low workloads, such as simple extraction or exact matching, operations of high workloads are run using multiple cores. Hence, it is critical to have efficient parallel execution and load balancing. Matching operations require data partitioning to facilitate parallel execution. Other operations, such as motif extraction, require the decomposition of the problem search space across cores. Either way, the number of cores to use is optimized to increase utilization and to speedup computation.

There is a trade-off between index size and computation. We combine indexing techniques with well-known algorithms (e.g. Boyer-Moore algorithm) to strike a balance between preprocessing time, index size, and execution efficiency. We opt for a novel suffix trie index that does not maintain terminating symbols and string identifiers, unlike generalized suffix trees [6]. The suffix trie indexes all suffixes of all strings and retains the frequency of every path label by storing a single integer in every node. Each node represents a character, avoiding the need to scan strings to retrieve path labels. Most operations are answered by simply traversing suffix tries. In cases where we need to relate path labels to strings, parallel algorithms are used. Moreover, our data structure is easily decomposable in the form of multiple suffix tries for different collection subsets.

In order to utilize large infrastructures, it is critical to find the best decomposition and to accurately estimate runtimes. StarDB adopts our automatic tuning framework [8] to decide the problem decomposition and estimate serial and parallel runtimes. Random sample tasks are used to model the workload of different decompositions. The gamma probability density function is used to approximate the workload frequency distributions. A discrete event simulator is then used to simulate execution and estimate runtimes. By automatically tuning itself, StarDB meets user time and budget constraints while efficiently utilizing available resources.

Example 2: The archive analyzed in Example 1 is represented logically by one collection but indexed using suffix tries that fit in memory. StarDB first executes the repeated motifs operation in parallel. Since the motifs search space is a combinatorial tree, it is logically partitioned into many sub-trees. On a supercomputer, the parallel planner finds that 2,048 cores can be fully utilized given the query workload. The original archive is not accessed because suffix tries are annotated with counts. The resulting repeated motifs are also in the form of a suffix trie. Therefore, the common suffixes are easily extracted by a simple index traversal.

3. DESCRIPTION OF DEMONSTRATION

This demonstration illustrates the power of StarDB in analyzing strings. The following features are shown:

1. Scalability to large infrastructures using real datasets.
2. Automatic tuning and elasticity features.
3. Perform analytical tasks online and in batch mode.
4. Ability to express complex string queries declaratively.

3.1 Datasets

In this demonstration, we use real datasets from bioinformatics and the English language. We choose these datasets to exhibit a variety of alphabets, strings of different lengths, and collections of different sizes and character repetition distributions. The query workloads are high enough to demonstrate StarDB power in a conference setting. In addition, the used datasets are one to two orders of magnitude larger than previously reported results in string databases and parallel string operators. We have tested StarDB using real datasets in the order of gigabytes (e.g., the complete human genome of size 2.6GB) and synthetic datasets with up to 100 symbol alphabets.

3.2 Infrastructure

We use a dedicated high-end 480-core Linux cluster for this demonstration. This cluster consists of 20 machines; each is equipped with 148GB RAM shared by 24 cores. We will also utilize 10 instances from Amazon EC2. StarDB fully utilizes the resource of an IBM Blue Gene/P supercomputer (16,384 CPUs) at over 90% speedup efficiency [10] but resource usage policies prevent us from using the supercomputer in a demo setting.

3.3 Scenarios and Workloads

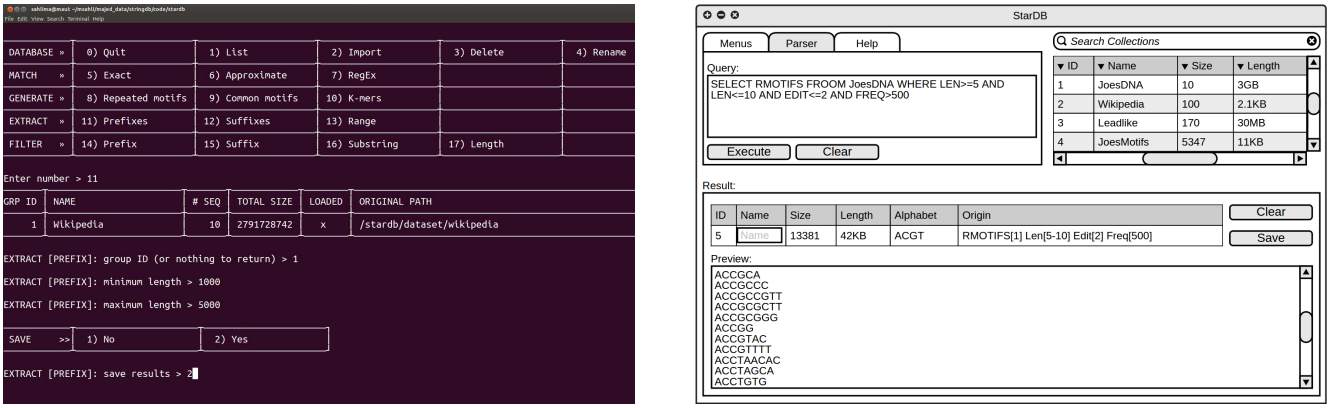
We provide a user interface to build tasks that get translated to StarQL queries (See Figure 3). We consider scenarios of users performing analytical tasks on strings; such as finding frequent patterns, matching, counting, and generating k-mers. Queries can be executed in different orders, nested, saved, and further processed. We showcase two different scenarios and discuss their workloads.

3.3.1 *Bioinformatics scenario*

Given DNA and protein datasets, a biologist needs to find the patterns that are frequent within every genomic sequence and at the same time common between sequences. Such patterns have potential functional importance and can be used to draw conclusions across species or to find mutations within a species. The workload of such task is high due to the combinatorial search space over the string's alphabet. The task translates to generating repeated motifs, generating common motifs, then finding the motifs that are both repeated and common. The following StarQL query is an example for this scenario.

```
SELECT CMOTIFS(dna) AS c
  FROM dna,(SELECT RMOTIFS AS r FROM dna
              WHERE r.LEN>10 AND r.EDIT<2
                AND r.FREQ>10000) AS rep
  WHERE c.LEN>10 AND c.EDIT<2 AND c.FREQ=10
    AND c.EXACT(rep);
```
(5)

First, the inner sub-query is evaluated and its results are saved as a collection (`rep`). This evaluation is done in parallel by executing the repeated motifs operator. The collection `rep` is only available for the current query because it is not imported. Then, the outer sub-query is evaluated by running the common motifs operator resulting in another temporary collection (`c`). The exact filter (`c.EXACT(rep)`) is used to find the intersection between the repeated and common motifs. After returning the final results, the temporary collections and their indexes are dropped.



(a) Command line interface (CLI)

(b) Graphical user interface (GUI)

Figure 3: Sample StarDB interface screens.

3.3.2 *Literary scenario*

Given the English text of Wikipedia, a librarian is curious about how writers start articles. She wants to explore the letters that frequently appear consecutively in the prefaces of Wikipedia pages. In StarDB, the k-mers operation finds symbols that appear consecutively and reports their frequencies. To find k-mers from the beginnings of articles, the k-mers are generated from the prefixes of the texts or are checked to exist in a specific range. To explore, the librarian may need to increase k-mers lengths as long as frequency is high. Given StarDB indexing, these queries are of low workloads. The following StarQL query is an example of the top 20 k-mers of length 4, 8, or 16, that appear in the first 100 characters more than 200 times across the dataset.

```
IMPORT (SELECT PREFIXES AS p FROM wiki
          WHERE p.LEN=100) AS pref;
SELECT KMERS AS k FROM pref WHERE k.LEN=4          (6)
  OR k.LEN=8 OR k.LEN=16 AND COUNT(k)>200
  ORDER BY COUNT(k) DESC LIMIT 20;
```

The first query translates to creating a new collection in the database from the results of the prefix extractor. The second query is run by executing the k-mers operation with length and frequency filters to keep the top 20 results only.

4. EXPERIMENTAL EVALUATION

We compare StarDB capabilities against state-of-the-art procedural repeated motif extractors. Table 1 below shows that StarDB is superior in terms of machine and data scalability. Although the procedural methods are specialized, StarDB generates the same output up to 3 orders of magnitude faster. For example, for a certain exact-length motif query, FLAME runs for 4 hours while StarDB finishes serially in 1 hour and using 12 cores in 7 minutes.

Table 1: Largest reported datasets and number of cores for repeated motif extraction.

System	Dataset Size	# Cores	Motif Types
FLAME [4]	1.3 MB	1	Exact-length
VARUN [1]	3.1 MB	1	Maximal
PSMILE [3]	0.2 MB	4	Exact-length
StarDB [ours]	2.6 GB	16,386	Any + Supermaximal

5. ACKNOWLEDGMENT

For computer time, this research used the resources of the Supercomputing Laboratory at King Abdullah University of Science & Technology (KAUST) in Thuwal, Saudi Arabia.

6. REFERENCES

[1] A. Apostolico, M. Comin, and L. Parida. VARUN: discovering extensible motifs under saturation constraints. *IEEE/ACM Tran. on Computational Biology Bioinformatics*, 7(4), 2010.

[2] J. A. Blake, C. J. Bult, et al. Beyond the data deluge: data integration and bio-ontologies. *Journal of biomedical informatics*, 39(3), 2006.

[3] A. M. Carvalho, A. L. Oliveira, A. T. Freitas, and M.-F. Sagot. A parallel algorithm for the extraction of structured motifs. In *Proc. of SAC*, 2004.

[4] A. Floratou, S. Tata, and J. M. Patel. Efficient and Accurate Discovery of Patterns in Sequence Data Sets. *TKDE*, 23(8), Aug. 2011.

[5] I. Gorton and D. K. Gracio. *Data-intensive computing: architectures, algorithms, and applications*. Cambridge University Press, 2012.

[6] D. Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology*. 1997.

[7] A. Mathur, A. Sihag, E. Bagaria, S. Rajawat, et al. A new perspective to data processing: Big data. In *Proc. of INDIACom*.

[8] M. Sahli, E. Mansour, T. Alturkestani, and P. Kalnis. Automatic tuning of bag-of-tasks application. In *Proc. of ICDE*, April 2015.

[9] M. Sahli, E. Mansour, and P. Kalnis. Parallel motif extraction from very long sequences. In *Proc. of CIKM*.

[10] M. Sahli, E. Mansour, and P. Kalnis. Acme: A scalable parallel system for extracting frequent patterns from a very long sequence. *The VLDBJ*, 23(6), Dec. 2014.

[11] S. Tata, W. Lang, and J. M. Patel. Periscope/SQ: Interactive exploration of biological sequence databases. In *Proc. of VLDB*, 2007.