Shared Bioinformatics Databases within the Unipro UGENE Platform

Ivan V. Protsyuk [1,2*], German A. Grekhov [1], Alexey V. Tiunov [1] and Mikhail Yu. Fursov [1]

[1]Unipro Center for Information Technologies, 6/1 Lavrentieva Ave., 630090 Novosibirsk, Russia

[2]Department of Physics, Novosibirsk State University, 2 Pirogova St., 630090 Novosibirsk, Russia

Summary

Unipro UGENE is an open-source bioinformatics toolkit that integrates popular tools along with original instruments for molecular biologists within a unified user interface. Nowadays, most bioinformatics desktop applications, including UGENE, make use of a local data model while processing different types of data. Such an approach causes an inconvenience for scientists working cooperatively and relying on the same data. This refers to the need of making multiple copies of certain files for every workplace and maintaining synchronization between them in case of modifications. Therefore, we focused on delivering a collaborative work into the UGENE user experience. Currently, several UGENE installations can be connected to a designated shared database and users can interact with it simultaneously. Such databases can be created by UGENE users and be used at their discretion. Objects of each data type, supported by UGENE such as sequences, annotations, multiple alignments, etc., can now be easily imported from or exported to a remote storage. One of the main advantages of this system, compared to existing ones, is the almost simultaneous access of client applications to shared data regardless of their volume. Moreover, the system is capable of storing millions of objects. The storage itself is a regular database server so even an inexpert user is able to deploy it. Thus, UGENE may provide access to shared data for users located, for example, in the same laboratory or institution. UGENE is available at: http://ugene.net/download.html.

1 Introduction

One of the major challenges in bioinformatics is the development of methods and algorithms for processing large data arrays that appear as a result of biological sequencing experiments. Average information flows produced by sequencing constitute hundreds of gigabytes per day; therefore, the issue of their reasonable storing is especially problematic. In order to solve the major part of biological data processing tasks, researchers utilize desktop bioinformatics applications offering required means and algorithms that provide an automated analysis of this information. However, these applications normally use a local data model, i.e. allow the processing of the information that is located immediately on a user's computer only. At the same time, today's scientific activity often requires many scientists to analyse the same data simultaneously. The use of desktop applications causes certain problems in this case. Firstly, the information needed is to be copied to every workplace, which can be a time-consuming task in itself due to large data volumes. Secondly, if one of several users modifies shared data, these changes have to be distributed to the other users, and this, again, requires multiple copy operations. However, several proprietary bioinformatics applications offer some solutions for the collaborative biological data access problem, and they will be discussed in further detail in Section 2.

* To whom correspondence should be addressed. Email: iprotsyuk@unipro.ru

The UGENE platform [1], within which this study was performed, represents one of the most commonly used software packages for carrying out various types of biological data analyses. UGENE provides a wide range of features including different types of visualization and workflow data processing. It is available on different operating systems since UGENE is being developed mainly with the use of the cross-platform Qt framework [1]. It is also provided free of charge, and its source code is distributed under the terms of GNU General Public License, v. 2.0 [3].

The aim of this study is to develop the shared biological data storage system within the UGENE platform. This will allow regular UGENE users to create their own shared databases for providing access to their genomic datasets for other scientists. Reaching this goal implies not only solving the collaborative data access problem, but also integrating the storage with visualization capabilities and algorithms available in UGENE. Eventually, it will help users save time.

In this work, the general architecture of the shared biological data storage, offered by the Geneious package [4], was used along with the concept of a public database server. At the same time, the approach to storing and arranging heterogeneous data was completely rethought. Certain data models were developed for each type of biological information supported by the UGENE platform such as sequences, multiple sequence alignments, genome assemblies, etc. In analogous systems, these pieces of data are serialized and stored uniformly (in file systems or databases) and loaded entirely to the client side, which is not applicable in case of large data volumes. Instead, the common idea of the data models, developed in this study for different types of biological data, involves the deep separation of the data and their loading to the client side piecemeal. Such an approach results in a shortened response time of the client application as well as a reduced network traffic in the system.

The paper is organized as follows. Section 2 gives an overview of related work. Details of the methods and approaches used in this work are outlined in Section 3. Results and discussion are addressed in Section 4. Finally, Section 5 summarizes the achievements of this study, the work currently in progress, and future plans.

2 Related work

Existing desktop systems for shared biological information storage generally realize two approaches. The shared database, and the shared data server. The first approach requires a public database server to be deployed by a user. Then, client applications can connect to it and store the information that, afterwards, becomes available for all the users of the system who are connected to that database. The shared database approach is then implemented, for example, in the Geneious application, developed by the BioMatters Company [4]. Then there is the shared data server approach which, besides database server, also includes the deployment of a web-server offered by the vendor of the system. In this case, all the requests to the database pass through the web-server. Thereby, the whole storage, in contrast to a stand-alone database server, becomes active, i.e. not only can it respond to client requests, but it also can send notifications to clients. Besides, it is not necessary to use a database as a storage engine. An ordinary file system suits this purpose well, since all incoming requests are processed by the web-server anyway. Another advantage is that the system can be accessed via a web-browser, as well as the client desktop application. This point increases the usability of the storage system. On the other hand, a web-server installation is a significantly more complicated procedure than a database server installation and it requires a qualified IT specialist to get involved. The shared data server technique is used by the CLC Bioinformatics Database application, offered by the CLC bio Company [5].

However, existing software leverages a unified approach to the data storage. Namely, all the biological objects (sequences, multiple alignments, etc.) are immediately stored in database records or in a server's file system as serialized data. Such an implementation requires a client to completely download data to the local host in case of any need of access to them, because the serialized state does not enable a fast information search. Therefore, this approach increases system response times when large objects, occupying several gigabytes, are involved. That is why within this study, we decided to implement the data storage in a database, using the structured data separation according to the constitution of each data type and typical operations applied to it. Such a method increases the upload time of information to the database, since additional ordering needs to be done, but reduces the time needed for subsequent access to it. Thereby, each client, accessing the storage, does not have to download a DNA sequence with the size of one gigabyte to his/her computer in order to visualize its region with a length of 1,000 nucleotides. Instead, one has only to download 1 KB of the information that describes that region. This simple example shows that, in case of large data arrays, the method of their separation and internal structuring can generally be more effective from the user's point of view than the unified storage approach.

3 Methods and approaches

3.1 Initial assumptions

In this work, the shared system was implemented using the approach of the shared database, described in the previous part. It enables even a non-technically educated user to deploy the storage for one's own purposes within a laboratory. Besides the simplicity from the user's point of view, the shared database approach has no essential drawbacks, compared to the shared data server, when it comes to the problem of collaborative data access. If necessary, this system may be extended with a web-server in the future in order to provide access to it via a web-browser.

Then, we chose a relational database as an underlying infrastructure of the shared storage. The most obvious reason for this is that relational databases are well known and widely used within all areas of database applications. Therefore, using this approach, our shared system is assumed to pose fewer additional challenges connected with tuning and administration of the database server itself. On the other hand, it is evident that modern NoSQL solutions can prove better horizontal scalability, fault tolerance and lower development costs, than their conventional SQL counterparts. The horizontal scalability, in turn, can reduce significantly the hardware costs of computational nodes constituting the shared storage. However, the use of such database providers is still quite rare in enterprise bioinformatics systems, so this can cause additional difficulties during deployment and maintenance. Besides, the considerable performance growth, compared to relational databases, is usually achieved only when several computational nodes are involved into a NoSQL system, whereas a single computer equipped with large-capacity hard disks is usually sufficient for the data storage in the majority of biological laboratories. For these reasons, we decided to develop relational data models instead of the NoSQL ones. Nonetheless, the architecture of the interaction between a client application and a shared storage, devised within this work, abstracts from the implementation of a particular underlying database. Instead, we have designed certain interfaces referenced by the client application that must be supported by the storage engine. Therefore, in the future it is possible to develop the implementation of these interfaces that is based on a NoSQL database. However, our first step is done towards the relational approach and the MySQL DBMS [6].

3.2 Storage of different data types

In this section, we consider a software architecture of the system in detail. Originally, the UGENE package includes the entities of a document and an object. The document represents some data resource and contains a list of objects. Objects, in turn, represent biological data of a certain type. The diagram, describing relations between these entities, is depicted on the Figure 1. For illustration purposes, it shows a few implementations of the object entity for biological sequences, multiple sequence alignments and annotations. In addition, there is a document format (see "DocumentFormat" on the Figure 1), i.e. the entity that initializes document and fills it with objects.

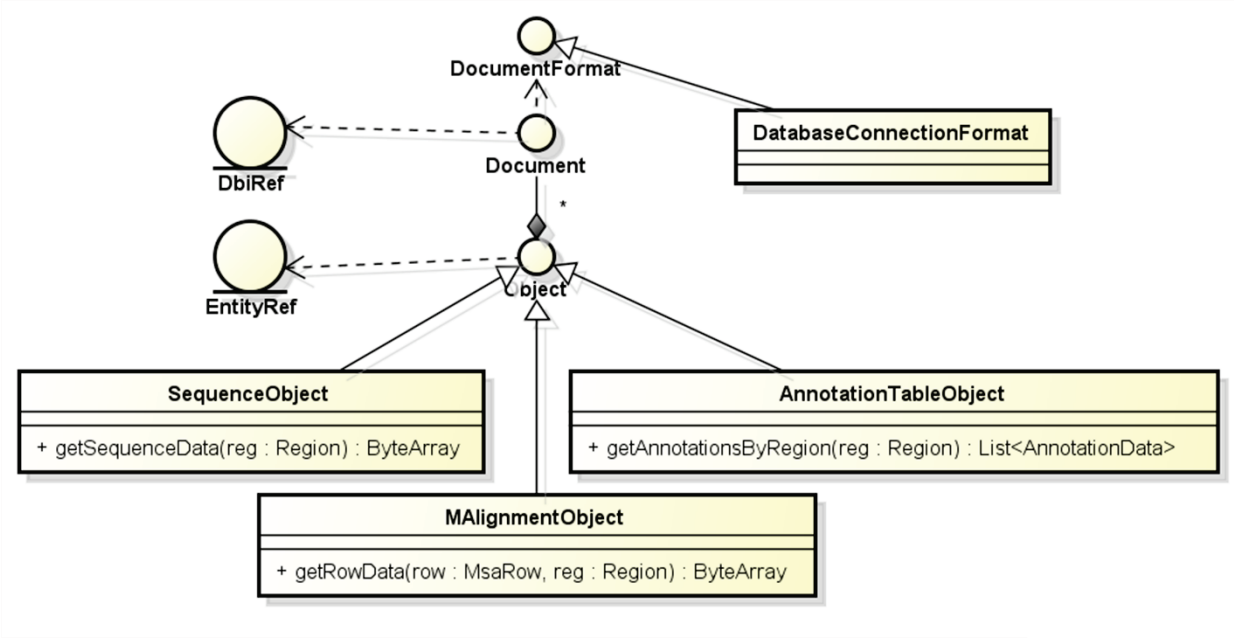

Figure 1: UGENE data sources diagram. The key entity here is "Document" containing a set of "Object" instances that can belong to different data types. Sequences, multiple sequence alignments and annotations are given as an example. The document has a reference to a database, where its objects are stored. At the same time, each object has a reference to particular records in the database. "Document format" is an entity that determines a document initialization procedure and encapsulates low-level input/output operations. Different document formats exist for various file formats and databases.

In case of the shared system, we decided to use abstractions within this scheme and implement one more specialization of "DocumentFormat" for creating a document based on an existing database (see "DatabaseConnectionFormat" on the Figure 1). This approach allows reusing the entire infrastructure, developed for document handling, for shared databases. Thus, from the user's point of view a shared database looks like a usual file from a local computer, possibly, containing a number of objects.

Each object belonging to a document has some reference (see "EntityRef" on the Figure 1) to relation records in a database where its content is stored. In fact, it is a numerical identifier. Therefore, it is not a resource-consuming procedure to download this information even for a large set of objects from the database. It is also important that such an index ("EntityRef") has a small size that does not depend on the amount of physical space occupied by a corresponding object. Thus, a client application can download a shallow copy of a shared database content that includes only numerical identifier of each object and their text names to be displayed for a user. All the further data are downloaded only upon a user's request to an object, such as visualization or saving to a local file.

Now let us consider directly the methods of storing different types of biological data in the database. First of all, it is substantial that not all of them may principally have large sizes (~1 GB) or complex internal structure. Some of them do not exceed several megabytes and are usually used entirely by biologists. Therefore, they can be stored similarly to how it is done in analogous software, namely, in serialized form in a single relation record of a database. In case of the access from the client side, this serialized data can be transferred over the network, and its objective representation will be created by the client application. Due to small object sizes, this approach does not cause problems with the performance of the system. Such data types are called "simple" in further.

For each of other object types a special data model needs to be developed that would correspond to their structure, typical operations applied to them and permit their separation to relatively small pieces for downloading them to the client side upon request. We call these data types "complex".

Within this study, all the types of the biological information, supported by the UGENE platform, were separated into the two categories defined above. The following types were referred to the simple ones: phylogenetic trees, chromatograms, tertiary structures of proteins, weight matrixes, frequency matrices and text files. A single independent piece of data belonging to any of these data types has a size no more than several megabytes in the overwhelming majority of cases. Here, "text files" refer to comments or remarks made by biologists or generated by sequencing hardware. They are also have relatively small sizes.

The complex data types include sequences (nucleotide and protein), annotations, multiple sequence alignments, genome assemblies and genetic variations.

Let us now consider the data models used for storing different types of biological information. We decided to store simple data types as serialized data in BLOB fields of database relations. Serialization is achieved by the conversion of an object data to its binary representation. Particular serialization techniques are out of the scope of this paper. However, they can be found in the public source code repository of the UGENE project.

For such complex type as a sequence BLOB fields have been also used, however, an entire object is not stored in a single field. Instead, it is separated uniformly into parts with a size of 1,000 bytes, and each of them is represented by a single BLOB field in the database. Accordingly, when a client accesses a sequence it specifies the particular region of the sequence and a selection is done in the database that includes records intersecting or constituting the region. On the Figure 2, relations "Sequence" and "SequenceData" are intended for the sequence storage. "SequenceData" is the very relation that contains 1,000 bytes sequence chunks along with their coordinates within a sequence. The "Sequence" relation, in turn, keeps some general information about a sequence such as its length (for caching purposes), an alphabet and a flag indicating whether the sequence is circular or not.

Additional research is needed to determine an optimal chunk size for the sequence partitioning. Obviously, it depends on a hardware configuration of a database server where storage is located. Large chunk sizes can bring about the additional memory consumption of a database server, while small ones increase the average time of data import to the database, since more queries have to be issued to the database. We decided to keep a relatively small chunk size in order not to deteriorate the performance of databases run on commodity computers. Probably, future UGENE versions will allow tuning the sequence partitioning for advanced users or will offer certain means for automatic detection of the optimal partitioning.

Such compound data types as multiple sequence alignments or genome assemblies are represented effectively by a set of sequences, so the approach, described above, for the storage of sequences was leveraged for them. However, there are additional data structures specific for each of these types. Namely, a multiple alignment is represented by an ordered list of sequences stored the same way as independent sequences described above. Nevertheless, alignments usually contain gaps inside, so supplementary data structures, describing relations between gaps and sequences were devised. Database relations "Msa", "MsaRow" and "MsaRowGap", shown on the Figure 2, are involved in the multiple alignment storage as a superstructure above a plain sequence list. Each alignment object has the corresponding record in the "Msa" relation that holds general cached information about the alignment such as its total length, the number of sequences and their alphabet. Aligned sequences, in turn, are described by records in the "MsaRow" relation that references both "Msa" and "Sequence" relations and determine absolute coordinates of a sequence in an alignment both vertically ("pos" field), i.e. a number of a sequence, and horizontally ("gstart" and "gend" fields) – sequence start and sequence end positions within an alignment. Finally, alignment gaps are stored in the "MsaRowGap" relation where each record references the "Msa" relation as well as the "MsaRow" relation and keeps absolute gap start and gap end positions for a particular alignment row.

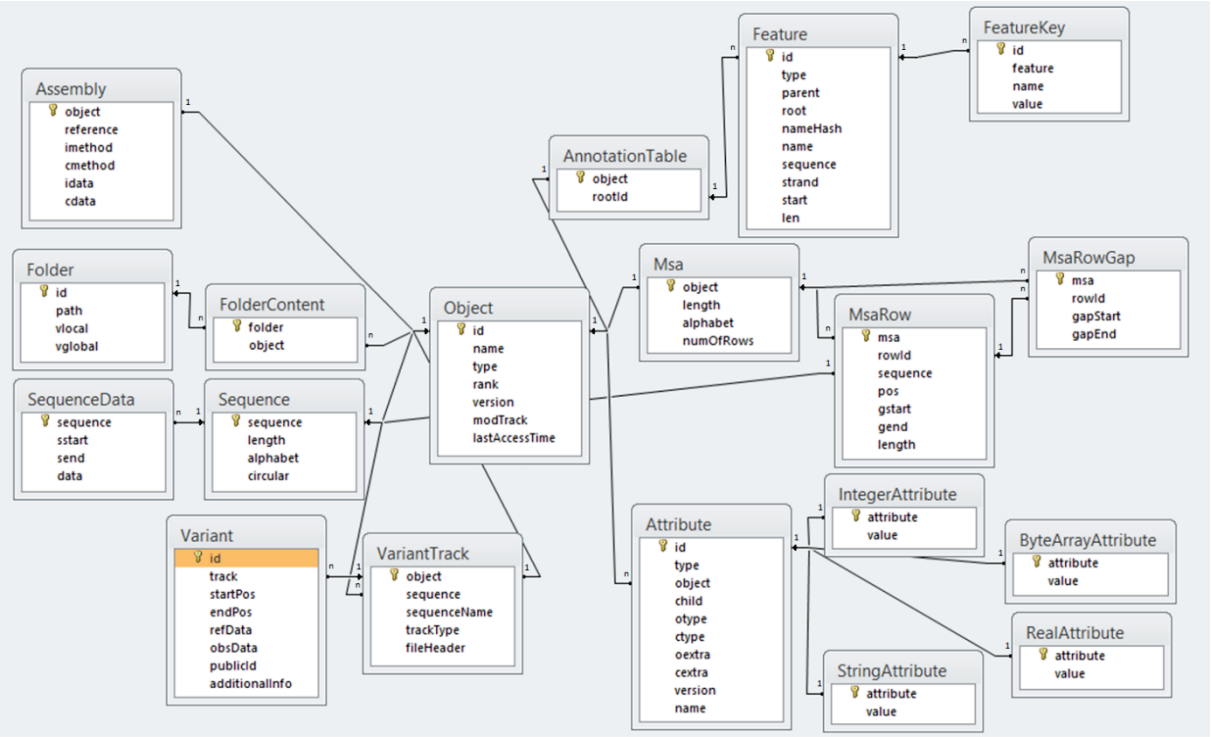

Figure 2: ER-diagram of the shared data storage for "complex" data types. The central entity here is a record in the "Object" relation. It serves for the storage of common attributes of biological objects including name and data type. Depending on a particular object type, its data are stored in different relations. For example, sequence objects are stored in one way, whereas annotations (or features) are considered to be separate objects and stored in another way. In addition, there are relations for arbitrary key-value attributes of any object type (for example, an accession number for a sequence) – common "Attribute" relation and a few value-dependent relations that keep values for each record from the "Attribute" relation.

On the other hand, genome assemblies usually occupy much more space than multiple alignments so another method is required for their storage that could support fast lookups for visualization. In this work, we decided to divide short reads, composing an assembly, into groups according to their length. Boundaries of the short read length are the following: 50, 100, 200, 400, 800, $4 \cdot 10^3$, $2.5 \cdot 10^4$, 10^5, $5 \cdot 10^5$ and $2 \cdot 10^6$. The non-uniformity of this sampling

is caused by that the overwhelming majority of short reads, obtained during sequencing, are of small sizes, no more than a couple of hundred base pairs. Therefore, they are to be separated at a lower scale than relatively long reads that constitute a small part of overall reads. Each group is stored within an isolated relation of the database. Such an approach proves better performance in case of assembly rendering than short read storage within a single database relation. The reason is that we index only short read start positions and, during visualization, process reads from only a few relations depending on a visualized assembly region. In addition, a total size of relation indexes is smaller in this case.

Let us now consider the annotation storage in the database. We treat an annotation as an emphasized region or a group of regions of a biological sequence that has a finite number of attributes represented by a set of key-value pairs, so called, annotation qualifiers. Annotations are joined into an annotation table (this term is not connected with database tables) which means that they all annotate a certain sequence. Within the software architecture of UGENE, annotations are incorporated into groups according to their biological meaning. For example, annotations that refer to the regions coding gene or containing a transcription factor binding site belong to different groups. Groups can be nested as well. Due to such a structure, we decided to store annotations in the database as n-ary trees. Each tree describes an annotation table and has a dummy root node that is unique for each tree. The “FeatureKey” relation (see Figure 2) stores annotation qualifiers, and the “Feature” relation accumulates annotations and annotation groups. Its “parent” column contains references to the parent node in a tree. An annotation has an annotation group as a parent, and a parent for an annotation group is either other group or a top-level root group. For caching purposes, the “root” column is introduced that contains references to a root node for each tree node. This improves the performance of an annotation selection within a particular annotation table. Certain columns, such as “start” or “len” representing an annotated region start position and its length respectively, make sense only for annotation storage and are not used for annotation groups. This overhead is acceptable because groups usually constitute less than 0.1% of all relation records.

Genomic variations describe mutations in a genome. Basically, they are simplified annotations because they always correspond to a single sequence region. Additionally, they are not incorporated into groups and lack qualifiers. Accordingly, an approach, similar to the one that was advised for annotations, was applied to store them. The “VariantTrack” relation from the Figure 2 holds records representing variant tracks, i.e. a set of variations for a particular sequence. Variations themselves are stored in the “Variant” relation that comprises variation coordinates (the “startPos” and “endPos” columns) and variation data – the “refData” and “obsData” columns containing a reference sequence and a mutated one respectively.

The data model developed within this work additionally provides the mean of a native data ordering. We developed an analog of a conventional file system directory tree for arranging objects in a shared database in folders from the UGENE GUI, as it is done on desktop computers. For the convenience of users, a special predefined folder “Recycle bin” was introduced. Normally, all other folders and objects deleted by a user appear in “Recycle bin” first, and only after that, they can be removed permanently. The information about folders and locations of different objects is stored in relations “Folder” and “FolderContent” (see Figure 2). Each record in the “Folder” relation corresponds to a single folder and contains its absolute path that is, in fact, a merged set of all parent folder names separated by a special symbol. The “FolderContent” relation provides links between objects and their parent folders.

4 Results

During initial tests of the developed system, the content of such public databases, widely used among scientific community, as PDB [7] and GenBank [8] has been successfully uploaded to it. The first one has size about 40 GB in a plain files form and contains about 10^5 objects that include sequences, annotations and tertiary protein structures. The second one is about 280 GB and contains over 10^6 objects that include sequences and corresponding annotations. Single objects of a large size have been supported as well. For example, a genome assembly of the Denisovan [9], comprising about 40 GB of short reads, has been successfully uploaded to the system. All the data uploaded to the system become available for reading and writing for other clients connected to it via the UGENE package.

The graphical representation for exploring and filtering shared database content has been developed and integrated into the existing GUI. Besides, client applications connected to a particular database check periodically, if other users have changed the database content somehow, and update the database content view appropriately.

Additionally, shared database capabilities were integrated with the UGENE Workflow Designer [1]. It is a workflow management system that is a part of the UGENE platform. Namely, a workflow can use shared data as input, and workflow computation results can be stored to a shared database. Since the Workflow Designer used to be a desktop tool before shared databases were introduced, it worked only with local files. Currently, this functionality is more versatile: workflows can process both local files and remote data from shared databases simultaneously. Different workflow writing elements within the same workflow can store workflow results to a local file system as well as shared databases.

As an example of a shared database instance, we have deployed a public database server with read-only access available for every UGENE user [10]. UGENE packages version 1.14.0 and higher are provided with predefined authentication information for this database. Currently, it comprises plasmid sequences downloaded from the UniVec database [11] and widely used genome sequences. There are chromosome sequences of mouse (mm9), human (hg19), drosophila (dm3) to name a few. All sequences have attached text files with particular links to their original locations in public sources. A detailed use case for connection and interaction with the public database can be found in the UGENE documentation [10].

Note that since the public storage server is physically located in Germany, the interaction with the public database goes on slightly slower than it does with a local database instance. A delay during sequence visualization is about 0.5 seconds depending on a client geographical location, whereas it totals tenths of a second in a local environment. This is caused mainly by a longer round-trip time between the client and the server. The delay deteriorates transmission speed especially in case of the fragmentation of server response packages, i.e. when their size exceeds MTU (maximum transmission unit) of a network path used for the connection.

A user manual describing the interaction with shared databases in detail, from questions of configuring and deployment to the procedures of storing and retrieving data, has been published online [12].

5 Conclusion

As a result of this study, the shared system for storing biological information has been designed and implemented. The shared database functionality is included into regular UGENE packages, and this allows every UGENE user to deploy one's own instance of a shared storage and use it for data storage and sharing with other users. A public shared database instance was installed and made available for the life science community. It contains

genomic datasets that are often used by biologists. Generally, the system is capable of storing millions of objects belonging to various data types. This makes possible using the UGENE package for shared data management within a biological laboratory or research institute. The first version of UGENE supporting shared databases is 1.14.0, and it was released on August 1, 2014.

Data models devised in this study cover all the data manipulations and data types supported by the UGENE platform. However, other projects aimed at the development of a comprehensive relational data model for genomic data exist, for example, GMOD Chado [13], BioSQL [14] and BioMart [15]. Although, they may provide access to specific data types that are not supported by UGENE, certain interfaces to such external systems can be integrated into future versions of UGENE in order to achieve interoperability with existing data storages. Two main issues need to be solved for this. The first one is the identification of entities that can be analyzed with UGENE in each of external databases. The second problem is that existing systems do not support directly partial sequence data fetching unlike the data storage described in this paper. The main reason for this is that the whole sequence data is stored in a single database record in the existing schemas. Database streaming technique can help overcome this impediment, but it is not supported natively by all database providers, for example, MySQL. Moreover, the use of streaming makes a database server performance the matter of concern in case of large sequences being involved because streamed data can be loaded completely to the memory of the server depending on a storage engine implementation. This is a potential bottleneck for a shared system with several users working simultaneously. The bottom line is certain workarounds and trade-offs have to be devised to gain the conformity with other data sources and provide their seamless integration into the UGENE user interface.

Our current work is focused on a series of optimizations for the work with a large number of shared objects. In this regard, the main disadvantage of the system is that the client application constantly monitors the state of all the objects in the database, and the amount of memory consumed by the client application grows linearly depending on a number of shared objects. It is required to develop the approach implying tracking only those objects that are in a user's field of view.

A drawback of the shared system in its current implementation is that it has lower fault tolerance, compared to the platforms considered in the Section 2. Since certain biological objects are never downloaded completely to the client side, an unexpected database server fault, occurred during computations, can lead to the loss of computed results. The solution for such situations is shared data caching by a client application. For example, local relational DBMSs, such as SQLite [16], can be used to recreate the shared storage data model and backup seamlessly the content of objects that are in use. Applying this technique, computations can be accomplished even after the database server is down.

Presently, objects in a shared database are read-only, i.e. they cannot be modified the same way as local files opened by UGENE. For example, changing a multiple alignment by inserting gaps cannot be done for a shared multiple alignment object. The support of shared data explicit modification is currently postponed since the major part of data in bioinformatics is rarely altered, but used for the production of some derived information instead. If necessary, this functionality can be added in future UGENE versions. In the current implementation, a possible workaround is the following: shared data can be exported to a client computer, then modified locally and uploaded again. The original objects may be deleted after that.

As for our long-term goals, the emergence of shared databases opens the opportunity of major advances for the Workflow Designer. Namely, if a shared database was incorporated with a

high-performance computational cluster, it would principally allow distributed processing the shared data using workflows. Being accessible via the Internet, such a system could provide a cloud service for biological data storage and processing. Thus, the development of the shared storage is our first step towards the high-performance computational bioinformatics system that does not require any other software except the UGENE platform.

The software is freely available (license: GPL) at: http://ugene.net/download.html.

Acknowledgements

This work was supported by the UNIPRO Center for Information Technologies (Novosibirsk, Russia). Our appreciation goes to other members of the Unipro UGENE project team for their ideas, discussions and help: Olga Golosova, Yuriy Vaskin, Denis Kandrov, Artyom Savchenko, Eugenia Pushkova, Kirill Rasputin, Vladimir Malinovsky, Yuliya Algaer, Alexey Varlamov, Timur Tleukenov and Konstantin Okonechnikov.

References

[1] K. Okonechnikov, O. Golosova, M. Fursov, the UGENE team. Unipro UGENE: a unified bioinformatics toolkit. *Bioinformatics*, 28(8): 1166-1167, 2012.

[2] Qt. Cross Platform Application & UI Development Framework. The Qt Company. [Online]. URL: http://www.qt.io. Accessed: 01-Jun-2015.

[3] GNU General Public License v2.0, GNU Project, Free Software Foundation, Inc. [Online]. URL: http://www.gnu.org/licenses/old-licenses/gpl-2.0.html. Accessed: 01-Jun-2015.

[4] M. Kearse, R. Moir, A. Wilson, S. Stones-Havas, M. Cheung, Sh. Sturrock, S. Buxton, A. Cooper, S. Markowitz, Ch. Duran, T. Thierer, B. Ashton, P. Meintjes, A. Drummond. Geneious Basic: An integrated and extendable desktop software platform for the organization and analysis of sequence data. *Bioinformatics*, 28(12): 1647-1649, 2012.

[5] CLC Bioinformatics Database. CLC bio, a QIAGEN Company. [Online]. URL: http://www.clcbio.com/products/clc-bioinformatics-database. Accessed: 01-Jun-2015.

[6] MySQL. The world's most popular open source database. [Online]. URL: http://www.mysql.com. Accessed: 01-Jun-2015.

[7] RCSB Protein Data Bank, Research Collaboratory for Structural Bioinformatics. MySQL. The world's most popular open source database. [Online]. URL: http://www.rcsb.org/pdb/home/home.do. Accessed: 01-Jun-2015.

[8] GenBank Home, National Center for Biotechnology Information, U.S. National Library of Medicine. [Online]. URL: http://www.ncbi.nlm.nih.gov/genbank. Accessed: 01-Jun-2015.

[9] J. Krause, Q. Fu, J. Good, B. Viola, M. Shunkov, A. Derevianko & S. Pääbo. The complete mitochondrial DNA genome of an unknown hominin from southern Siberia. *Nature*, 464 (7290): 894–897, 2012.

[10] UGENE Public Storage – Unipro UGENE Online User Manual v.1.14.0 – WIKI. [Online]. URL: http://ugene.net/wiki/display/UUOUM/UGENE+Public+Storage. Accessed: 01-Jun-2015.

[11] The UniVec Database, National Center for Biotechnology Information, U.S. National Library of Medicine. [Online]. URL: http://www.ncbi.nlm.nih.gov/tools/vecscreen/univec/. Accessed: 01-Jun-2015.

[12] Shared Database – Unipro UGENE Online User Manual v.1.14.0 – WIKI. [Online]. URL: http://ugene.net/wiki/display/UUOUM/Shared+Database. Accessed: 01-Jun-2015.
[13] C. Mungall, D. Emmert, the FlyBase Consortium. A Chado case study: an ontology-based modular schema for representing genome-associated biological information. *Bioinformatics*, 23(13): 337-346, 2007.
[14] BioSQL. Open Bioinformatics Foundation. [Online]. URL: https://www.biosql.org/wiki/Main_Page. Accessed: 01-Jun-2015.
[15] A. Kasprzyk. BioMart: driving a paradigm change in biological data management. *Database: The Journal of Biological Databases and Curation*, 2011: bar049, 2011.
[16] SQLite Home Page. [Online]. URL: http://sqlite.org. Accessed: 01-Jun-2015.