Bioinformation by Biomedical Informatics Publishing Group *open access*

www.bioinformation.net **Database**

Synchronization of APIIS based farm animal biodiversity systems

Zhivko Duchev[1, 2*] and Eildert Groeneveld[2]

[1]Department of Animal Breeding and Genetics, University of Veterinary Medicine, 30559 Hannover, Germany; [2]Institute for Animal Breeding, Federal Agricultural Research Center (FAL), 31535 Neustadt, Germany;

Zhivko Duchev* - Email: duchev@tzv.fal.de; * Corresponding author

received March 24, 2006; revised April 25, 2006; accepted April 30, 2006; published online May 03, 2006

Abstract:

One of the major problems in the management of farm animal and biodiversity information is the exchange of data and keeping it up-to-date, an issue that is very common with distributed information systems consisting of number of databases. This article describes the synchronization protocol developed in APIIS (adaptable platform independent information system) framework and reviews the basic considerations required when building distributed information system that has to exchange information in a network of APIIS based systems. The protocol is designed to synchronize a common part of different database structures. It is developed without any intended use of proprietary database engine and can work with a variety of RDBMS (relational database management system). The main targets of the protocol are animal biodiversity information systems without permanently connected nodes. The EFABIS (European farm animal biodiversity information system) is reviewed as an example of the implementation.

Availability: The synchronization protocol is integrated as a part of the APIIS framework, which is freely available from the authors.

Keywords: database management; data synchronization; farm animal information systems

Glossary of Terms:

Data element (DE) = smallest amount of data treated as one block in the synchronization process. DE is the list of columns from a defined subset of records in a table; Node = each independent part (database) of the global information network; Source = any node that distributes data elements to other nodes; Target = set of nodes to which one source distributes a data element; network manager = the management authority that will route the traffic of information, preventing conflicts or inconsistencies.

Background:

Development of open source databases in the APIIS framework **[1]** is common and the installation of identical systems becomes a financially affordable option. As outlined in elsewhere **[2],** data collection in animal agriculture relies on the distributed collection of farm animal data: these may originate on many farms, on test stations and in laboratories. These different sources will have to be integrated into one central database for across herd evaluations as is typically done in Best Linear Unbiased Prediction (BLUP) genetic evaluation **[3]** in selection programs. In principle, peripheral databases can be viewed as subsets of the central system both in terms of the database structure and also in terms of business rules. Clearly, business rules should be enforced at the initial data entry where - in case of errors - the original information is close at hand for correction. These local herd systems may be copies of the central system expanded in scope for on herd management as is done in commercial herd management packages which are available for nearly all species in animal agriculture. With this topology the task of transferring data from the periphery (e.g. farms) to the center is the last step in building a comprehensive central database.

Because all business rules have already been enforced at the periphery using a set identical to the rules at the central database, a generic data transfer can be employed which amounts to the synchronization among the central and the peripheral databases, without a need to consider business rules at this stage. The EFABIS network has a similar topology: there is a world wide central node of the biodiversity database at the United Nations FAO (Food and Agricultural Organization) in Rome. **[4]** On the regional level the EAAP (European Association of Animal productions) runs a database **[5]** with expanded information both in terms of content and structure, while countries like Poland have their own national database comprising all information from the levels above plus additional national data not to be shared with the other levels. Again: data is collected at the national and perhaps for some countries at the regional and worldwide level and will have to be propagated to all other levels. In line with the example from the animal agriculture, we have an identical core structure of the databases at all levels and are enforcing the same set of business rules everywhere. Thus, after initial data entry - at whatever level - information newly added to the database must be transferred, i.e. can be synchronized with the other databases in the network. Development, implementation and performance of such a synchronization procedure are described here.

Methodlogy:

Synchronization Requirements

The synchronization requirements were derived according to the requirements of EFABIS network.

Requirement #1

Each DE has a primary copy and there is only one database in the network where it can be edited. In animal breeding information systems, data is usually collected on different places like artificial insemination stations, farms and research institutions. All these sources of information keep copies of

data, there is someone (human or organization) who is officially responsible for the quality of data and all users of these data rely on its representative value. As an example the veterinary examination of the animal can be taken and there is a paper document containing the animal identification data, the veterinarian data and the results. Therefore, a natural requirement is that each data element should have a primary copy at one node where this element can be changed. This is the node where the person collecting the DE always enters the data, and in all other nodes this data will be read-only. This requirement ensures also a clear responsibility for the accuracy and up-to-date status of each DE. For example, each country in EFABIS that presents its data to the European (EAAP) and global (FAO) level is responsible for the data quality and consistency.

Requirement #2
For each DE the "distribution target" (nodes that want to obtain this element) is defined. In general terms, the data collection process does not end in itself. Usually the collected data is intended to be used by someone and in most cases the data users are clearly defined. For example, data collected on testing stations may be sent to a research institute for calculating the breeding values and the results are returned to the farmers. Very often there is a strictly defined hierarchy in the system with one central database collecting all data as a data-warehouse. This is the situation in the EFABIS network, where each European country sends data to EAAP and EAAP distributes part of the data to FAO. Therefore, for each data element there must be a well defined target group of nodes which needs this DE. This set of nodes is actually the “distribution target”. It could be also empty if this element is only for local use and will not be propagated.

Requirement #3
Each DE to be included in the synchronization process has to be defined by both source and target nodes. The DE to be transferred has to be negotiated and approved by the both sides. When a reconciliation session is started it automatically synchronizes all approved DE, thus not allowing the user to refuse the changes. This principle looks very restricting, but follows from the requirement of primary copy. The idea behind this requirement is that a user who needs a certain DE is accepting by default all changes, relying on the fact that they are representative. For example, if the primary copy of DE is deleted, then this element should be deleted everywhere. In contrast, the act of removing DE from the synchronization list has to be confirmed by both sides. Distribution sources and targets may be changed as long as this does not produce inconsistencies. This principle ensures that each node can choose the source and target nodes for a DE, unless this will disturb normal flow of data in the network. This implies that all changes in the DE path have to be coordinated by all nodes that exchange this DE.

Requirement #4
Each node can distribute all public data elements loaded in its database. If the primary copy node is the only source for a DE then this will produce a bottleneck in the data-flow. Therefore, each node that has received a public DE as a result of a synchronization process should be also allowed to propagate it further. This is not the case with non-public data elements and such elements can be distributed only to a subset of authorized nodes.

Requirement #5
Synchronization should not require human intervention. The protocol should be completely automated and be able to run on a regular basis as a scheduled task. It should not produce any inconsistencies in the target node, because such discrepancies usually require human intervention from scientific and technical persons - the former to solve the conflict, the latter to introduce the changes to the database. The process of solving conflicts is time consuming, and it requires the original data copy.

Requirement #6
A network regulating mechanism for the data-flow should exist. As the nodes are equal in rights and part of the requirements rely on the negotiation between two nodes, an unregulated data-flow can produce locks in the system. Therefore, if the network has no inbuilt "by design" clear data-flow, it has to be regulated by set of rules. They will prevent actions that are against the system consistency or resolve data-exchange conflicts between the nodes. The need for such rules can be seen from the following example: Let the node A target one of its data elements to node B and node B target this DE to node C. Let also presume that by system design node B has to have always this DE. In this situation if node A wants to change the target of this DE to node C then node B will loose its source. There are two possible solutions of this conflict: Node A is not allowed to change the target, because it will produce inconsistency. It is allowed to change the target, but has to do this in cooperation with node C, which will target it to node B.

Requirement #7
The system is loosely coupled and not all nodes are connected all the time. Although the access to the INTERNET is getting cheaper, there are a lot of farms, even in European countries, where the only option for connection is via phone line or satellite. An example is PISSA (Pig Information System South Africa) where data are collected in the farms off-line and then sent to the center once per week via e-mail.

Requirement #8
The protocol should be able to synchronize data over LAN (local area network) and WAN (wide area networks) such as INTERNET. It has to ensure secure transfer of the data over the public parts of the network. The nodes of the animal information system which uses this protocol can be part of the internal network of one organization or can be connected via INTERNET. Therefore, the synchronization protocol should use network transport protocols which are applicable everywhere. And as the data exchanged can be private, the protocol has to encrypt it when transferring over a public network.

Requirement #9
The protocol has to be able to exchange text and binary data. The last but not least requirement is related to the type of

information exchanged. We will not only synchronize data fields in the database containing quantitative values like size, milk, wool length, but also documents and multimedia data. This may look obvious, but it is important for the type and quantity of the data that will be transmitted.

Analysis of the requirements and description of the developed synchronization protocol

Analysis of the requirements

The requirement for one node where a user can change a DE puts us in situation similar to the Lazy Master Replication model from distributed systems. **[6].**According to this model, when the user updates a DE, only the primary copy of this DE in its master node is updated. Then, in separate transactions the master node updates each replica. There is a certain time of inconsistency between the master node and the replicas. Therefore this model is called lazy or asynchronous. In our system we also use the primary copy approach, but we presume that not all nodes can connect to the master node. Therefore, the propagation of changes to the other nodes is done in a cascading manner and the nodes using the master node as a source are updated first, then their target nodes are updated and so on until all nodes are updated. Each node except the master one will be in an asynchronous state until one of its sources is updated and synchronization with that source take place. To assess the time in asynchronous state and its impact on the system functionality we have to look at the specificities of the information systems we are dealing with. In the national and supranational biodiversity systems like EFABIS a detailed breed description, morphology, performance and demographic data are collected. New data in such systems are loaded on a monthly or even on yearly basis. On the other hand, in herd management information systems, the central database receives data weekly or daily. Hence, synchronization per day [or] per week is sufficient.

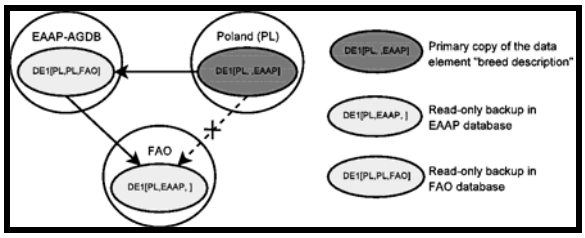

Figure 1: Excerpt of EFABIS topology, containing National Polish database, European regional one and the world database of FAO. The dashed arrow represents the rule for one source per data element

The other issue to be considered here is the number of sources a node can use for a DE. Having several sources for one DE can lead to values collisions - if node C gets two different DE versions from node A and node B. Such problems are discussed elsewhere. **[7, 8]** A simple restriction that solves such conflicts is the limitation of one source for each data element. Two different databases can have two different sources for the same DE, but a single database can have only one source for its DE. If the user node can establish a connection to more than one node keeping a DE then the user can choose, in accordance with the network rules, which one will be used as a source and also move from one source to another, but cannot use two sources simultaneously. The one-source requirement produces a bottleneck in the data-flow, because the possible interval to update DE is the intersection of the online time of the source and the target node. This restriction does not have big impact on the animal biodiversity databases, as already stated, the time interval between the updates is relatively long. In herd management animal information systems each DE comes to the central database usually from one source, e.g. field test data for a certain animal comes from one farm. Moreover with proper management of the network, there can be a scheduled interval of time when both nodes are online for synchronization.

In each node we have information about each data element's route. This can be fully described by the expression:

DE [Primary Copy, Source, Target]
Primary Copy :: = Node Name
Source :: = Node Name | Empty
Target :: = List of Node Names | Empty

'Primary Copy' is the name of the node where this DE was initially entered. The 'Source' is the node that has supplied this element and 'Target' is the list of nodes, this element will be delivered to. These expressions are illustrated in an example from EFABIS, shown in Figure 1. In this example we use the DE 'breed description', which includes the general description of all Polish breeds. This DE is initially entered in the Polish database. Poland distributes it to the EAAP database and from there it will be propagated to the world database of FAO. As a result, we have the following descriptions of the DE's route in the various databases. In the Polish node (named PL), it is described as DE[*PL*, , *EAAP*]. Here the 'Primary Copy' is PL because it is the first node where this DE is entered in the system. The 'Source' field is empty, since we have not received this DE as part of synchronization process. The 'Target' list consists of only one element – 'EAAP', because only the EAAP node will receive it directly from Poland. Following the same logic the description in the EAAP node is DE[*PL, PL, FAO*] and in the FAO's database DE[*PL, EAAP,*]. The empty Target field in the last description means that FAO will not distribute this DE to other nodes.

The source and target fields of a DE's route description in the nodes as defined above are sufficient to determine the route of DE within the whole network, resulting in a tree structure. The 'Primary Copy' node is the root while the sources and targets describe the ribs. Two checks have to be done in this structure: (1) the ribs definition is consistent, and (2) there are no cycles in the tree. The former check can be done for each of the two nodes such that Node1 (source) and Node2 (target) exchange the element DE1:

(Node2.DE1[Source]=Node1) and (Node2 in Node1.DE1[Target])

The main problem with both checks is that information is spread around the nodes and has to be collected in one place by the manager of the network.

There are two possible types of synchronization: (1) based on a log(journal) and (2) based on a state. The journal synchronization is based on the logging of all data modification statements that are executed on the source database and sending them to target node. It is suitable when there is a lot of data with relatively small number of changes. The log-based reconciliation is used in systems like replicated dictionary **[9],** Bayou **[10],** Vagabond **[11]** and StorageBox. **[12]** The general problem with this approach is that the node has to ensure that all targets have updated their state before removing the statement from the log file, which can lead to accumulating large amounts of unused data. This can happen for example, if a node which is in the target list does never connect to the source node. The approaches to this issue vary from discarding writes from logs in **[10]** to removing long latent target nodes from the replication set **[13],** but they are not suitable in our setup.

Therefore, the alternative is synchronization 'by state' - based on version vectors or time-stamping of the data elements. **[14, 15]** The time-stamp approach requires clock synchronization as shown in **[16],** which is practically impossible in the network of independent databases like EFABIS. Therefore, the versioning approach was chosen, where each record has an integer version attached, which is incremented on update. In the synchronization session, the source node plays the role of the server and the target node that of a client. The client sends the current version number of the DE to the server where it is compared with the server's own version. If the server's version is new the updated DE is propagated to the client. The shortcoming of this method is that each time the versions of all DEs to be synchronized are compared, thus making the overhead proportional to the number of records. This approach is suitable for databases with relatively small number of records and this is the case with farm animal biodiversity databases collecting cumulated data on breed level. For example, the European regional database and Polish National database in EFABIS contain 21,4426 and 7,290 records respectively.

Database structure

To execute and manage synchronization between databases the following additions are made to the database structure:

Additional columns

It is well known from the replicated databases that each record has an identifier that is unique within the information network. To ensure this independently from the RDBMS engine, which is one requirement of the APIIS design **[**Error! Reference source not found.**],** a new 'system' column for the Global Unique Identifier 'guid' has to be added to each table. Upon insertion in the primary copy node the 'guid' is automatically set from a sequence. On the other hand the synchronization process has to preserve the 'guid' in the target database.

The synchronization 'by state' requires to keep track of the changes made to records. Therefore, in each table an additional field for the record version has to be added. When initially inserted in the primary copy node all records from one DE have a version set to one and each update increments the version by one. This field should be also included in the synchronization of a DE.

By definition, each DE consists of "defined subset of records". Such classification in APIIS is done on the basis of the additional 'class' column which has been added to each table. It is up to the designer to define the classes when designing the system. The classification of the records can be done on the basis of the location where the original data were collected. In IS collecting individual animal data, these places can be farms, breeding societies, test stations. As an example let we have a system collecting herdbook data for animals from three farms. Then each record can be classified as 'Farm1', 'Farm2' or 'Farm3', depending on the farm of the animal.

To have option for private data in the database, each record has a Boolean 'synch' field. The flag stored in this field is used to indicate if the record is targeted for synchronization. The user when entering data explicitly sets this flag. Examples of the meta-fields are shown in Figure 2.

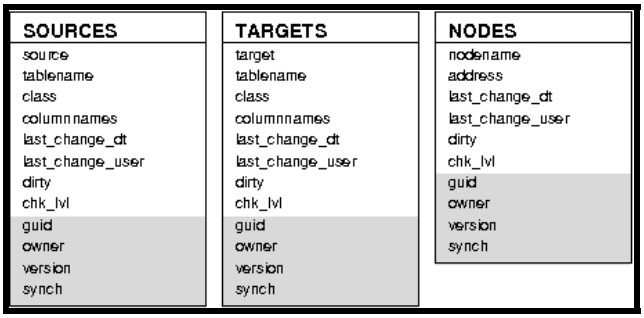

Figure 2: Additional tables and meta-fields (shaded) needed for the synchronization protocol in the APIIS structure

Additional tables:

The management of routes requires three 'system' tables in each database. The table 'Nodes' contains the names of the nodes and their physical IP-addresses. Each node which is source or target of the current node must be registered in this table. All names must be unique within the network with each IP address linked only to one node. The other two tables – 'Sources' and 'Targets' (Figure 2) are used for specifying the incoming and outgoing data elements and their nodes through the ('columnnames', 'class', 'tablename', 'source'|'target') columns.

Synchronization protocol

The synchronization protocol is of client-server type. Each node, which distributes data elements, has a server daemon listening for incoming connections. Such a node will be referred in the following as 'server'. On the other side, the node which wants to update its data from the server is the "client" and has to run the client part of the software. When a connection is initiated by the client, it starts with a handshaking to verify if the server is free for synchronization. In this case, the client reads the description of the first DE, from the server and sends this description called DED (Data Element Description) for confirmation. After successful confirmation from the server, the client reads the state of this DE (the guid

and version of all records described by the DE) and sends this information to the other side. Then the server compares this information with its own state and chooses appropriate action for updating the client:

```
foreach client.record in DE1 {
  if (not exists server.record) then
    client.record.action='Delete';
}
foreach server.record in DE1 {
  if (not exists client.record) then
    client.record.action='Insert';
  else if(server.record.version>client.record.version) then
        client.record.action='Update';
}
```

The action and the data retrieved from the server (in case of insert or update) are encapsulated into a merge structure and send back to the client. There the merge structure is transformed in SQL statements in the client's native SQL dialect and the database is updated. The functional model of the synchronization process for one DE is shown in Figure 3. The same steps are repeated for all other DE expected from the server. The entire operation is treated as one transaction and changes are committed only if all DE are successfully updated. This is a weakness of the protocol, because in case of error, the synchronization has to be started from the beginning. The block diagram of the used algorithm is shown in Figure 4.

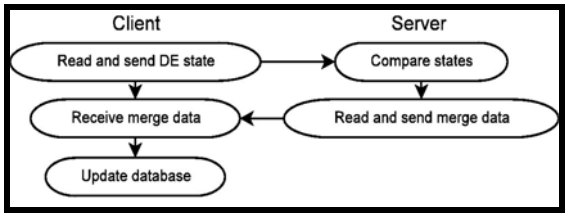

Figure 3: Functional model of the data synchronization process for a single data element

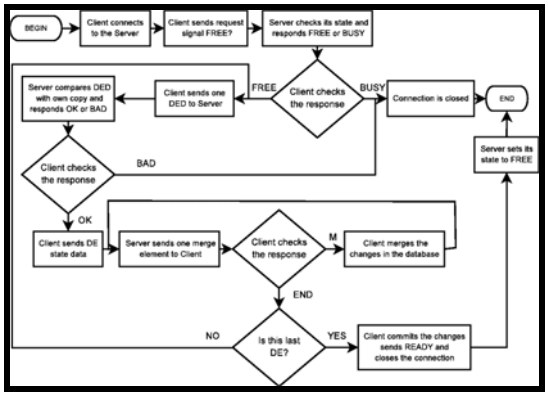

Figure 4: Block diagram of the synchronization algorithm

Implementation in EFABIS

The automated synchronization protocol was implemented in EFABIS (European Farm Animal Biodiversity Information System). This system is a network of databases collecting biodiversity data from European countries and transfers it to the central European database. The European database will be used as a data source for the world biodiversity database of FAO, especially for the data that is required to build the World Watch List for endangered domestic breeds. **[17]** The data collected in EFABIS describes the farm animal breeds in terms of naming, origin and development, morphology, special qualities, performance, demographic trends and conservation programs.

In EFABIS we have clear hierarchy on three levels - National, Regional and World level. Each lower level is an expansion of the previous one in terms of content and structure. For example the National Polish Biodiversity database **[18]** stores data required by EAAP and additional data for species like fish and small fury animals, which are not represented at the European level. The Polish database has the structure of the European one plus extensions in terms of additional tables and fields, to handle the country specific data. The flow of data (as shown in Figure 5) is bi-directional. National databases have to send the data required by the European database and from there a subset will be sent to the World database. The world database will also propagate some data to the national databases, e.g. documents, images and common codes that have to be uniform within the whole network like codes for the species and sex. Such codes must be introduced only in one place, i.e. - the FAO database and accepted from the other nodes. Each node has also the option of having private codes, but these have to be used only in private data which will be not synchronized.

Each national node is named by the ISO-3166-1-alpha-2 code elements of the International Organization for Standardization **[19]**, i.e. BG for Bulgaria, PL for Poland. This ensures unique name even within the whole world. Two exceptions are the names of the European database and the global database - for the former 'EAAP' is used, and for the latter - 'FAO'. To ensure uniqueness of the 'guids' and all other internal identifiers each database has to have a separate range for the sequence generators. Operationally, this was done by attaching to the official list of countries a predefined range for each country. This range should be set in the configuration files needed for creating a new node. All other actions required are completely automated by the software.

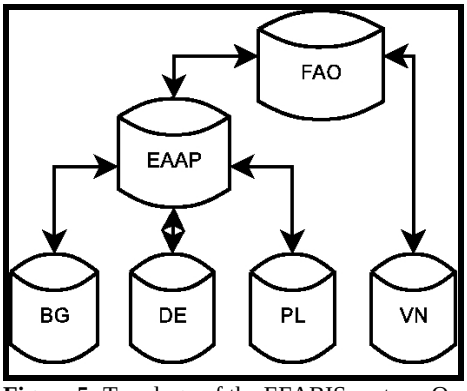

Figure 5: Topology of the EFABIS system. On the lower row the national databases are represented - BG for Bulgaria, DE - Germany, PL - Poland, VN - Viet Nam. In the middle row are the regional databases, e.g. EAAP for the Animal Genetic Data Bank of the European Association for Animal Production. On the top is the biodiversity database of FAO

Each record has to be identified as owned by a country or by one of the supranational databases. This give us the unique classification of the records and therefore the class column in EFABIS was named 'owner'. As a value the two-letter codes of the countries, along with 'EAAP' and 'FAO' are used. This marker is set automatically by the system through the country that a user is attached to. This setup allows countries that do not have their own database to load data directly in the European database. The EAAP node will be primary copy for these records, i.e. these data can be edited only there. The class column 'owner' was also used in the Access Control System of EFABIS, not only to mark the backups of DE as read-only, but also to define complete set of access rights for all users in the network.

For the transport part of the synchronization the TCP/IP protocol was used. The server daemon was set to listen on port 5433. If the node is behind firewall, it has to be set to allow incoming connections on this port. For compression and encryption of the transfer data, Mina Naguib's Net::EasyTCP module from CPAN **[20]** is used. The encryption of the data is done using the Blowfish algorithm. **[21]**

As an interesting byproduct the synchronization procedure has been used to initialize newly created national databases. By simply defining appropriate sources and targets, the national data was downloaded in one-time transfer from the global database.

Performance

Two nodes were established on two different machines for testing the synchronization protocol. One of the machines was with Mobile Intel(R) Pentium(R) 4, CPU 3.06GHz, 512MB RAM and the other with 4 64 bit AMD Opteron™ 850, CPU 2.4GHz, 8GB RAM. The tests were done using three types of connections: 100Mb/sec, 10Mb/sec and 128Kb/sec. In each test the time to receive merge data from the server, the time to update the database and the total synchronization time was measured. The results, rounded to the next integer are shown in Table 1. The value of 100,000 merge records was chosen to test the situation of initial loading of the database, while the other runs represent the amount of the updates expected in small and middle databases. The amount of memory used for the state information for 100,000 records was 5.54 MB, or 59 bytes per record. This information was read from the database in 2.35 sec and transmitted to the client in 1.128 sec using a 10 Mb/sec connection. As can be seen from the results with fast connections, the time needed to transfer data over the network is approximately the same or even less than the time needed to merge changes in the client database, i.e. the bottleneck is the communication with the database backend. Another interesting observation is the time required to receive data on 100Mb/sec and 10Mb/sec were practically the same. This means that the protocol does not use the whole bandwidth which is also confirmed by the results on 128 Kb/sec. When reducing the bandwidth 80 times (from 10 Mb/sec to 128 Kb/sec) we have only about fifteen fold increase of the receive time. A similar tendency can be observed in the total synchronization time - it increases only six time required from 100Mb/sec and 10 Mb/sec to 128 Kb/sec. This speed of the protocol is mainly due to the fact that it internally encrypts and compresses data for secure transfer over public network and usually the average record size in animal databases is relatively small - about 500 B. Therefore, the results on a 128Kb/sec connection can be considered sufficient for production systems with slow internet connections.

Time to receive merge data (seconds)				
Bandwidth / Merge records count	100	1000	10000	100000
100 Mb/sec	1	2	11	80
10 Mb/sec	1	2	13	82
128 Kb/sec	2	13	129	1279
Time to update the database (seconds)				
Bandwidth / Merge records count	100	1000	10000	100000
100 Mb/sec	1	2	17	180
10 Mb/sec	1	2	17	183
128 Kb/sec	1	2	16	163
Total synchronization time (seconds)				
Bandwidth / Merge records count	100	1000	10000	100000
100 Mb/sec	7	9	33	268
10 Mb/sec	8	10	36	270
128 Kb/sec	34	46	175	1451

Table 1: Time to receive merge data from the server, update the database and total synchronization time

Conclusions:

The synchronization protocol designed for data exchange between loosely coupled nodes in farm animal information systems relies on the strict primary copy approach for each data element, thus avoiding update conflicts and the need for human intervention. The drawback of the protocol is the transfer and comparison of the DE state information on each run. Nevertheless, the protocol shows good results on medium and small databases similar to the ones used in biodiversity, national gene banks and small population management information systems. The total synchronization time scales using a 128 Kb/sec connection, allows running the protocol as scheduled on a daily basis.

References:

[1] E. Groeneveld, *Livestock Prod Sci*., 87:1 (2004)
[2] E. Groeneveld, *Proc 7th WCGALP*, 33:721 (2002)
[3] C. R. Henderson, *Biometrics*, 31:423 (1975) [PMID:1174616]
[4] http://www.fao.org/dad-is/
[5] http://www.tihohannover.de/einricht/zucht/eaap/index.htm
[6] J. Gray, *et al., Proc SIGMOD '96*, 25:173 (1996)
[7] B. Douglas, *et al., Proc SOSP-15*, 172 (1995)
[8] T. Ekenstam, *et al., Distributed and Parallel Databases*, 9:187 (2001)
[9] G. T. J. Wuu & A. J. Bernstein, *ACM SIGOPS Oper. Syst. Rev.*, 20:57 (1986)

[10] K. Petersen, *et al., Proc SOSP-16*, 288 (1997)
[11] K. Nørvåg & K. Bratbergsengen, *DEXA Workshop*, 728 (1997)
[12] F. Hupfeld, *ICDCS Workshops*, 458 (2004)
[13] R. Ladin, *et al., ACM Trans. on Comp. Syst.*, 10: 360 (1992)
[14] D. S. Parker, *et al., IEEE Trans. Software Eng.*, 9:240 (1983)
[15] P. S. Almeida, *et al., Proc ICDCS-22*, 544 (2002)
[16] L. Lamport, *Commun. ACM*, 21:558 (1978)
[17] B. D. Scherf, *World Watch List for domestic animal diversity*, (2000)
[18] http://efabis.izoo.krakow.pl
[19] http://www.iso.org/
[20] http://www.cpan.org/
[21] B. Schneier, *Fast Software Encryption, Cambr. Sec. Workshop*, 191 (1994)

Edited by P. Kangueane
Citation: Duchev & Groeneveld, Bioinformation 1(5): 146-152 (2006)