Computer Science • Vol. 10 • 2009

Piotr Oramus[*]

PARALLEL AND DISTRIBUTED CALCULATIONS SUPPORTED AND MANAGED BY THE RELATIONAL DATABASE

A simple, based on a relational database, system, for a management of a parallel and a distributed computer calculations, is presented. In the proposed system, the parallel calculations are carried out according to a master/slave model. Because an input and an output data of programs are stored directly in the database, the use of files is reduced to a minimum. The management system allows for a combination of computing power of many computers for solving a single numerical problem.

Keywords: *parallel calculations, relational database, master/slave model*

RÓWNOLEGŁE I ROZPROSZONE OBLICZENIA WSPIERANE I ZARZĄDZANE ZA POMOCĄ RELACYJNEJ BAZY DANYCH

W pracy prezentowany jest prosty, oparty o relacyjną baze danych, system, przeznaczony do zarządzania równoległymi i rozposzonymi obliczeniami komputerowymi. W proponowanym systemie, obliczenia równoległe prowadzone są zgodnie z modelem master/slave. Ponieważ zarówno dane wejściowe, jak i wyjściowe programów zapisywane są bezpośrednio w bazie danych, użycie plików zostało ograniczone do minimum. System zarządzania pozwala na połączenie mocy obliczeniowej wielu komputerów w celu rozwiązania pojedynczego problemu numerycznego.

Słowa kluczowe: *obliczenia równoległe, relacyjna baza danych, model master/slave*

1. Introduction

Parallel computing plays an important role in solving large numerical problems. In this approach, all running programs need to closely collaborate in order to solve a common task, so communication between programs is inevitable.

One can use e.g. Message Passing Interface [1] to create parallel application, but to start calculations direct connection between all computers taking part in this job, is

[*] Department for Information Technology, Faculty of Physics, Astronomy and Applied Computer Science, Jagiellonian University, Krakow, Poland, `piotr.oramus@uj.edu.pl`

required. Unfortunately, in many cases, the condition of the direct communication can not be fulfilled. E.g. in the case of clusters, nodes typically do not have assigned public IP addresses and there is no direct access to them from the Internet. In addition, it is difficult to use several supercomputers in a single parallel calculation, because their users cannot start tasks on demand, but have to use a local batch system. Such a non homogeneous environment requires a special approach. Here, the database is proposed as a focal point for data exchange and tasks management.

When a client-server software architecture is considered (the use of the database is an example of such a case) all connections are initialized by the client programs in direction to the server, therefore the direct access from the Internet to all computers is not longer required – the access from all computers to the database server is sufficient.

Another and separate issue is an unpredictable time of such a connection. The clients can connect and disconnect at any time and a software, which is responsible for a job management, has to solve this problem.

A modern and well known system for a distributed computation is BOINC [2], which is a flexible and highly scalable platform for volunteer and desktop Grid computing. BOINC uses client-server architecture and the database to manage jobs, but the data flow and the storage model are based on files.

The intention of the author is to present a simple, single-user system, where information is stored only in the database, so Structured Query Language (SQL) can be directly used for a data analysis. Such a system may be easily and rapidly constructed and it allows to use for a parallel calculation a large variety of different computers.

2. Description of a problem solution

The relational database has repeatedly demonstrated its services to store and search large data sets. The combination of the database management system (DBMS) with an appropriate library allows to quickly build a program, which can retrieve all the necessary data from the database. When complete information for the tasks to be performed is stored in the database, the distributed computing system can easily be built – the only requirement is an availability of the network connection to the database.

This distributed system is capable of performing parallel calculations thanks to supplying additional software (a master program), which manages the task list on the basis of an analysis of already obtained results.

In the next section of this paper the relational database is presented. Then the description of the database based management system is given. Afterwards, a use of Specialized SQL Structures is shown. A software and a hardware used in the calculations are described in next section. The last section contains presentation of the system application to the problem of the construction of the optimal neural network ensemble by means of the parallel genetic algorithm.

3. The relational database

The database stores in a computer system a collection of data. Data is organized according to certain database model. In the case of the relational database [3] data is stored in two-dimensional tables called relations. A single relation usually contains multiple rows (called tuples), which have the same set of attributes (columns). To uniquely identify each row a so-called primary key is chosen as a single attribute or a set of attributes. The tuples can have the values of the primary key from another relation as its own attribute. A foreign key combines data saved in different relations. A structure of the database is described by means of a schema. The schema can be written in a simple form as follows: *relation_name* (attribute1_name, attribute2_name, etc.) and in this form will be used in this paper. The underline distinguishes the attribute, which plays the role of the primary key. To speed up search queries indexes can be created. One index can be composed of one or more attributes.

From a point of view of the database user, the most important issue is integrity of the data. DBMS can help the user to achieve this goal by means of a transaction and ability to lock tables. The transaction is an “all-or-nothing” mechanism: either all operations on data in tables are done correctly or nothing is changed. Usually the capability to serve multiple clients simultaneously is a wanted feature of DBMS, but in some cases this may lead to data corruption (e.g. when two users try to buy the same ticket). Locking tables can give a guarantee that two or more processes do not have an access to the data at the same time.

4. The management system based on the relational database

4.1. How not to use files

Usually at the beginning of an execution, a computer program reads its parameters from files. Then, results of calculations are saved to other files. Such a scheme of work has a large drawback: files need to be (usually manually) copied between systems taking part in the calculations and often exist in multiple, difficult to synchronize, copies. This makes both a management and a analysis of the results difficult.

A simple observation can help: typically, the input data and the results of the calculations take form of a two-dimensional table, which can be easily replaced by a relation in the database. In general there are two possibilities: one either creates relations designed for a certain set of data or uses general purpose relations having a scheme like *matrix_name* (row, col, value). The first solution is highly efficient and allows data selection and elaboration on the level of DBMS. Additionally, in this solution columns of one tuple (attributes) have meaningful names, and in the program source code can be accessed by those names (creation of Specialized SQL Structures (SSQLS), where a member variables correspond to attributes in the SQL relation, is easy and in the case of MySQL++ library requires only several lines of a source code [4] – an example of SSQLS use will be given later), so the probability of their wrong use is significantly lowered. The second solution requires additional relation e.g. *matrix_register* (matrix_id, matrix_name, number_of_rows, number_of_columns, description, time_stamp) to organize data and uses more disk space, but it has a large advantage: in a program all such data can be accessed and used in the same way with the help of one common piece of a source code (preferably a class in an object oriented language).

In the presented system "Matrix" class was developed. An object of this class represents a re-sizeable, two-dimensional array of floating point numbers with multiple mathematical operators defined. Such an object can be stored in the database in relation of the general purpose scheme. To support old programs, which still must use files, "Matrix" object can be saved/read in/from a text file.

4.2. Master/slave model

Master/slave model is a simple model, where one entity (master) controls what multiple other entities (slaves) do. This parallel programming paradigm is easy to implement, but has some important drawbacks such as a single point of failure[5].

In the presented system the main calculation tasks are done by "slave" programs. They are connected to the database and periodically check the *jobs* (job_id, experiment_id, parameters_relation_name, parameters_relation_id, status) relation. The attributes "parameters_relation_name" and "parameters_relation_id" mean here: the name of the relation containing the job initial parameters and a value of a primary key in that relation pointing at appropriate tuple, respectively.

When a job having a status set to "waiting" is found, the slave program locks the *jobs* relation to prevent simultaneous access from other programs, changes its status to "working" and unlocks the table. When work is completed, results are saved in the database and the status of the job is changed to "done", then the whole algorithm repeats. In the case of no work to be done, the slave program calls system sleep() function, so they almost do not consume system resources. The maximum execution time (real time) of the slave program is limited, so all the programs finally terminate even if the limit of the CPU time usage is not exceeded.

The *jobs* relation can be populated either manually (e.g. by means of phpMyAdmin web fronted to MySQL) or by another program (even in the form of a UNIX shell script). When the parallel execution is required, a master program can be written to control *jobs* relation.

To support multiple calculations carried out at the same time and to ease the management, relation *experiments* (experiment_id, description, parameters) was created. One master program controls one "experiment" and receives the value of experiment_id primary key from a command line. It must be emphasized, that a lot of management tasks done by the master program, are implemented by means of SQL queries and are executed directly by DBMS.

4.3. Cleaner program

To protect the calculation process from being stopped by even a single slave program, which fetched the job but was not able to complete it, the idea of a ”cleaner” program was developed. The idea is as follows: each running copy of the program has to register itself in the ***hosts*** (host_id, host_name, up_time, time_stamp) relation and periodically updates its own tuple. Thus, every program in the system possesses its own and unique identification obtained from the database (primary key host_id has ability to auto increment and its value is returned to the program by DBMS). The host_id attribute was added to the relation ***jobs***, relation ***experiments*** and all relations used for storing the results of the calculations. The host_id attribute has to be filled by the program changing tuples in those relations. Information about which program executes a certain job is now available in the system.

The cleaner program checks periodically the tuples in the ***hosts*** relation looking for an entry, which is not updated. When such a tuple is found, the cleaner reads host_id from this tuple, removes it from the ***hosts*** relation and changes the status of a job executed by the program with this host_id from “working” back to “waiting” state. A partial results, if any, returned by this program are removed in similar fashion.

The program, which cannot find its own tuple in the ***hosts*** relation is immediately terminated. This feature allows the termination of the program without a need of logging in on an appropriate computer and killing a program process. Additionally this feature can be used to automatically shut down the system when calculations are finished.

4.4. Specialized SQL Structures

Specialized SQL Structures are an important tool in developing programs, which have to communicate with the database. An access to data stored e.g. in the relation ***hosts*** (host_id, host_name, up_time), can be obtained by using the C++ structure having member variables corresponding to each attribute in this relation. Proper C++ structure, named here hosts, is defined indirectly by means of a short macro:

```
sql_create_4(hosts, 1, 4,
    mysqlpp::sql_int_unsigned, host_id,
    mysqlpp::sql_varchar, host_name,
    mysqlpp::sql_double, up_time )
```

The exact meaning of the macro parameters is described in details on the web page [6]. The same macro creates all methods required to exchange data with the database. Finally, to send a single record to the database, a small piece of C++ code is sufficient:

```
// First, connection to the database on a remote server has to be established
mysqlpp::Connection con_to_server( database, server_name, username, password);
// Here variable this_host is declared and initialized
hosts this_host( 1, "main_server", 1000.01 );
// We exchange data with the database by means of the class mysqlpp::Query
mysqlpp::Query query = con_to_server.query();
```

```
// The insert() method creates the SQL statement, appropriate for that case
query.insert( this_host );
// finally, query is executed
query.execute();
```

5. Software and hardware used

The slave programs were successfully installed on many computers having different hardware architectures. The standard PCs, one computing cluster (`mars.cyfronet.pl`) and two supercomputers (`panda.cyfronet.pl` and `baribal.cyfronet.pl`) were used together, connected to one database. The MySQL DBMS was installed on a desktop computer with a single Intel(R) Pentium(R) D processor running at 3.40GHz. The desktop computer was equipped with 8 GB of a RAM memory and a software RAID 5 disk array. This computer was also used to run the master and the cleaner programs.

All computers were running Linux operating system. The programs and all required libraries (Fast Artificial Neural Network and MySQL++) were compiled by means of GNU or Intel C/C++ compiler.

Figures presented in this paper were generated by means of simple R language scripts [7]. What must be emphasized, the results of the calculations were directly downloaded from the database to R language environment thanks to RMySQL package.

6. An application of the system to the construction of an artificial neural network ensemble

The Fast Artificial Neural Network library (FANN [8]) was used to create a classification program. Additional relations intended for a storage of the FANN parameters were created in the database. Training, validation and test patterns were also saved in the same database. The calculations presented in this chapter were preformed to test the management system under a heavy load. The total number of executed jobs exceed one and a half million. The stability, load and response time of the database server were checked.

6.1. Data preparation

Individual ANNs were concurrently trained on artificial set of data representing the probability for finding 26 letters (A-Z) in nine different human languages. The aim was to find a system able to correctly classify the language even if a text sample was short (like 260 characters). All pattern were generated by the script written in R language.

The tuples representing the training tasks were generated and saved in the *jobs* relation by means of simple shell script. After a training process, 480 different artificial neural networks were available for the next experiments.

6.2. An ensemble of the artificial neural networks

It is shown that the prediction ability of the artificial neural network ensemble is better than that of a single network [9] [10] [11]. The presented system was used to build such ensembles from an existing pool of ANN saved in the database. The size of the ensemble was limited to 15 networks. The output of the ensemble was calculated as a simple sum of the output of each network. The classification error of the ensemble was defined as follows:

$$Err = \frac{(0.9\cdot \mathit{number\ of\ wrongly\ classified\ patterns} + 0.1\cdot \frac{\mathit{number\ of\ bit\ fails}}{\mathit{number\ of\ outputs}})}{\mathit{number\ of\ patterns}}$$

The number of bit fails is defined as the number of the output neurons which differ from known output patterns more than 0.35.

6.3. The parallel genetic algorithm [12]

The configuration of the ensemble was encoded by means of bit string of length 135 positions (15 independent networks; one network identified by a group of 9 bits). The evaluation function for a chromosome was inversely proportional to the classification error of the ensemble coded by this chromosome:

$$Ev_i = 1/Err_i$$

where i is a unique chromosome identification number.

A fitness (measure of the reproduction opportunities) associated with the chromosome i was defined by:

$$F_i = Ev_i/\overline{Ev}$$

where $\overline{Ev}$ was the average evaluation of the population.

6.4. Calculations

The populations of the chromosomes were generated and stored in the database by the master program. The slave programs were responsible for a calculation of a value of the evaluation function. When all jobs for a current generation were completed, the fitness was computed by the master program. Then, an intermediate population was generated from a current population by the roulette-wheel selection [13]. The next generation was created by the recombination of parent strings taken in pairs from the intermediate generation. Afterwards, the mutation operator was applied. In all cases, two, the best chromosomes were directly copied from the previous to the next generation, so the best results were conserved.

6.5. Results

The populations of 100 and 300 chromosomes were used to optimize the ensemble of ANNs with respect to the classification error. Three different values of the mutation probability were used: 0.01, 0.02 and 0.05. One set of the parameters was used in three independent runs. The average results are given in Figure 1. The best results were achieved when the mutation probability was set to 0.01.

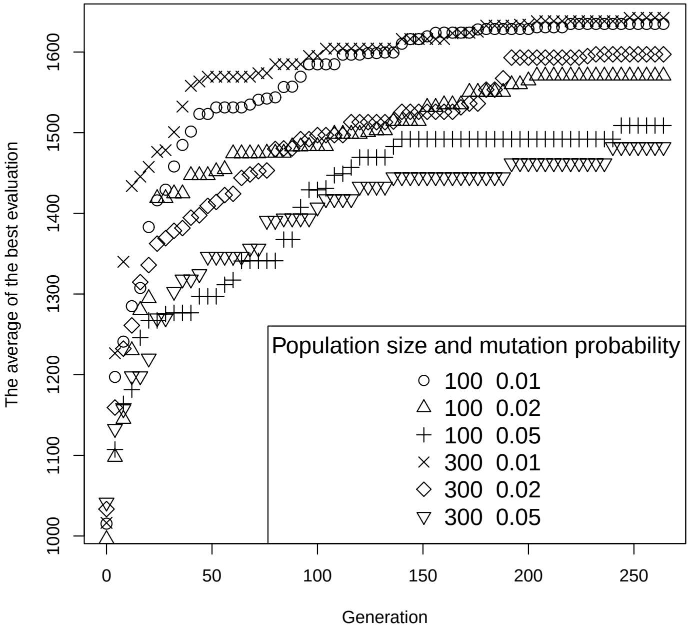

Fig. 1. The average of the best value of the evaluation function plotted versus generation number

In the next simulations the number of chromosomes in the intermediate generation was limited. Thus, the next generation was populated using the same chromosome from intermediate generation for several times. The calculations were performed for the population composed of 100 chromosomes and were repeated eight times for one set of the parameters. Results are plotted in Figure 2. The decrease of the intermediate generation size resulted in deterioration of the optimization ability.

In general, it was found out, that the ensemble of the neural networks used for the pattern classification was mistaken several times less frequently than the best single network.

7. Conclusions and future work

The presented system was able to effectively manage large scale calculations, in which more than 100 simultaneously running programs (18 master programs, one cleaner and 94 slaves) were used together. The system was remarkably stable. The cleaner program was always able to recognize interrupted tasks and to automatically "return" them back to the `jobs` queue.

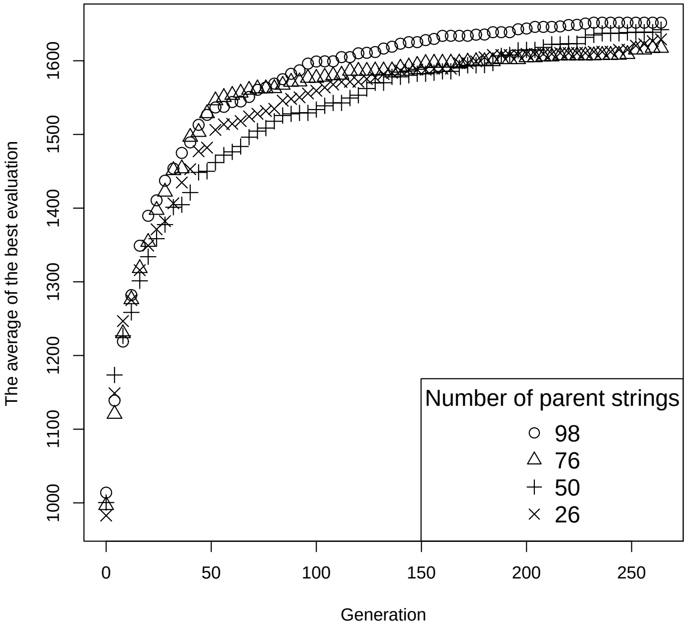

Fig. 2. The average of the best value of the evaluation function plotted versus generation number

The average use of CPU of the database server was around 70%.

Unfortunately, the increase in the number of rows in the database relations resulted in a significant slowdown in the search of the result data. In the future it is planned to generate a separate set of the relations to handle a single experiment.

Acknowledgements

The research reported here was partially supported by the grant no. MNiSW/SGI4700/UJ/126/2007.

References

[1] Pacheco P. S.: *A User's Guide to MPI.* University of San Francisco, 1998.

[2] `http://boinc.berkeley.edu/`

[3] Codd E. F.: *A Relational Model of Data for Large Shared Data Banks.* Communications of the ACM, vol. 13 (6), 1970, 377–387

[4] `http://tangentsoft.net/mysql++/doc/html/userman/ssqls.html`

[5] Sullivan M. P., Anderson D. P.: *Marionette: a System for Parallel Distributed Programming Using a Master/Slave Model.* EECS Department, University of Cali-

fornia, Berkeley, 1988 (`http://www.eecs.berkeley.edu/Pubs/TechRpts/1988/5728.html`)

[6] `http://tangentsoft.net/mysql++/doc/html/userman/`

[7] `http://www.r-project.org/`

[8] `http://leenissen.dk/fann/`

[9] Hansen L. K., Salomon P.: *Neural Network Ensembles*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 12, 1990, 993–1001

[10] Krogh A., Vedelsby J.: *Neural network ensembles, cross validation, and active learning*. In Advances in Neural Information Processing Systems, MIT, vol. 7, 1995, 231–238

[11] Sharkey A. J. C. : *On combining Artificial Naural Nets*. Connection Science, vol. 8, 3 / 4, 1996, 299-314

[12] Poli R., Langdon W.andB., McPhee N.andF., Koza J. R.: *A Field Guide to Genetic Programming* (`http://dces.essex.ac.uk/staff/rpoli/gp-field-guide/`)

[13] Whitley D.: *A Genetic Algorithm Tutorial* (`http://samizdat.mines.edu/ga_tutorial/`)