Modern Applied Science; Vol. 9, No. 4; 2015
ISSN 1913-1844 E-ISSN 1913-1852
Published by Canadian Center of Science and Education

Data Storage Control System Design

Yury Yurievich Shumilov[1] & Nikolay Sergeevich Dudakov[2]

[1] National Research Nuclear University 'MEPhI', Moscow, Russian Federation

[2] JSC 'Concern Systemprom', Moscow, Russian Federation

Correspondence: Yury Yurievich Shumilov, National Research Nuclear University 'MEPhI', Moscow, 115409, Russian Federation. E-mail: shumilovyy@gmail.com

Received: November 19, 2014 Accepted: November 27, 2014 Online Published: December 10, 2014

doi:10.5539/mas.v9n4p67 URL: http://dx.doi.org/10.5539/mas.v9n4p67

Abstract

The paper presents a methodology for evaluating and improving the effectiveness of storage management during the development of automated control systems. The description of the storage management system in terms of queuing theory is proposed. The model of the system and the criteria for efficient processing of requests to read and write data are provided. The authors also propose the partitioning of stored data and the use of several software solutions to improve the system performance.

Keywords: database design, modelling and management; database architectures; queuing theory; integer programming; constrained optimisation; nonlinear programming

1. Introduction

Nowadays automated control systems (ACS) are becoming more and more widespread. ACSs are computer-aided systems aimed to convert information, to make calculations and logical operations based on computer networks and modern information technology (IT). ACSs have proliferated in the form of control loops within manufacturing, transport, construction and other economic processes. The purpose of automated control systems is to ensure sound management of a complex object (process) in accordance with performance targets.

In the ACS design process one of the most important aspects is data access, storage and processing control. Functional and efficient ACS operation is impossible without IT appliances for data storage and processing. Demands to databases (data storage control system or DSCS), the core of modern automatic control systems, increase with growing data amounts and complexity of calculations (Wiegers, 2004).

2. ACS Data Storage Control System Design

All the applications and modules of the system, such as sources of information, analysis modules, ptimization modules, etc. one way or another get access to the stored information during ACS operation. DSCS (hereinafter also called database) is a central element in the information system, its operation characteristics, ease of access to the data affects the work of the entire complex. That means that in case of a large number of database accesses it may be the efficiency bottleneck of the complex. The efficiency of data storage is dependent on the software that accesses the data, the data model and performance of the equipment.

At the moment there are many data storage control systems that satisfy various needs. Besides, it is important that the majority of software solutions is in the public domain or based on open-source models (Ensor, Stevenson, 1999). Database design is a dynamic area of software development. A variety of problems set leads to designing databases that differ greatly by their characteristics and application purposes.

Most data systems are based on a client-server architecture, meaning 67ptimizatio data storage at the server and reading and writing access to the data by client applications. The advantages of client-server architectures are reliability, security, ease of maintenance, low load on client workstations. The disadvantages are big load on the server and high demands to the communication channels between the clients and the server reliability and capacity. The client-server architecture is most convenient to store large amounts of information but is rarely requested, as the frequent transfer of large amounts of data through the network can be difficult.

Most requests to the database server in modern information systems are read requests (documentation on 'Mirror Database Tools'). Therefore, read requests mainly determine the load on the server database and make the server and the network a client-server system 'bottleneck' in terms of its performance. Given low-flow characteristics

of data transmission channels and high server load in some cases, the shortcomings of the 'classic' client-server system significantly affect performance of the ACS.

An another approach involves storing copies of the database at each site (documentation on Oracle database). This solves some of the problems, and significantly reduces the load on the server because of a single transmission of a change to the connected sites. However, in this case the problem of 68ptimization68 the client and server copies of the database becomes much more difficult. Also, a client site either requests a full copy of the database when connecting to the server, or logging and 'partial update' mechanisms for site 68ptimization68n are necessary. In general, unlike the client-server architecture storage, databases with this architecture are more suitable for frequently requested data of relatively small volumes.

To demonstrate the above-described it can be noted that the cached systems greatly outperform the client-server ones in read rate, especially in the case of a growing number of reading applications (documentation on PostgreSQL database, (documentation on SQLite database). The dependence of the reading speed on a fixed number of objects for the PosgreSQL remote database and SQLite local database is represented in Figure 1:

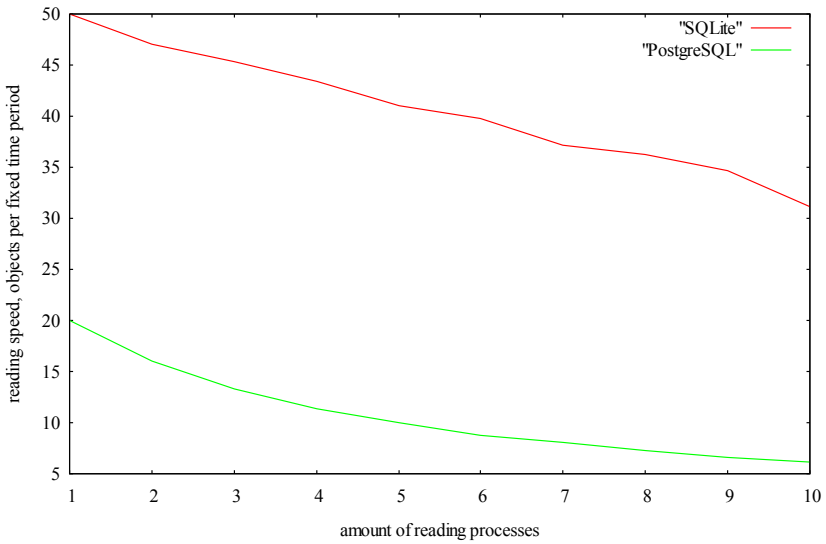

Figure 1. The dependence of the reading speed on the number of processes

A number of software solutions combine the features of both mentioned architectures implementing partial separation of the stored data and disposing part of it in local databases. In such cases the factor that determines performance of the system for fixed software solutions is the algorithm of data separation (the local part selection). In most such systems data separation is done by cashing – that is the dynamic placement of frequently requested data in the local database (documentation on memcached storage system). The cashing technique allows operation only with the frequency of requests and does not take into account the characteristics of local databases (if the local database differs from the main database). In some cases on-going data separation is based on the object domain and 'exemplary' properties of parts.

Another quite common approach is to design a distributed database in which data is shared between the servers in the cluster so that each server stores the data and the whole database allocation table, requests on pieces of data are transferred to the appropriate servers and the results are compiled afterwards (Carey, Livny, 1988). This approach has given a good account of itself, and there are a lot of projects in this area, both open (documentation on distributed system Apache Hadoop) and commercial (documentation on distributed system The Google File System). The main advantages of the approach are the reduction of the load on individual servers and the relative ease of scaling the data up to the volume of petabytes. The main disadvantages are complexity and concurrency control. According to well-known CAP-theorem, we can't achieve simultaneously all three guarantees of consistency, availability and partition tolerance. (Gilbert, Seth, Lynch, 2012)

The main difficulty of the data storage control system development is the heterogeneity of the stored data, which makes designing a universal internal storage engine (the DSCS engine) a complex task. There is, for example, a significant difference in access rates of the heterogeneous data. In ACSs the data types vary from the large

volume of relatively rarely changing objects to smaller volumes of data requested and modified multiple times per second. In some cases, special different precautions up to storage of the encrypted information may be required (Dudakov, Pirogov, Shumilov, 2009).

Considering the specific software solutions, it can be noted that in some cases performance of the database that proved to be faster during the simple operations dropped considerably with an increase in the intensity of the load. At the same time, software solutions specifically designed to be accessed by multiple applications are more resistant to competitive behavior.

In general, in accordance with the internal architecture of each software solution there is data with the 'most appropriate' set of properties, which is processed most effectively, and data with the 'less appropriate' set of properties, the processing of which can be difficult. At the same time, given the considerable diversity of the data stored, the stringent requirements and high load of modern ACSs, processed data characteristics cannot be neglected for the benefit of a software solution performance.

Thus, the lack of a universal open source database and the restriction on the power and cost of server hardware leads to the fact that when a large-scale ACS aimed at the processing of heterogeneous data is designed, it is often impossible to meet all the requirements by one ready-made software solution.

The partitioning of data and use of multiple interacting databases with modified software solutions could be a possible means of improving data storage efficiency. Accordingly, the stored information is considered as a set of data classes (tables in a relational data model) and the improvement of storage efficiency is proposed by means of a considered choice between ways to distribute the classes of data in databases. The choice of several software solutions allows us to combine the advantages of each system, avoiding, if possible, the shortcomings due to of the best-fit match of the data type and the respective database. The constancy of the partitioning ensures constant characteristics of the system if load characteristics also remain constant. That is not achieved by cashing.

During the DSCS design process a mathematical model of a query to storage processing (the system of several storages) was established. The model and the efficiency criteria can numerically assess the performance of one or more databases, which, in turn, enables an evaluation of the best way to distribute the data, the suitability and feasibility of a software solution.

The DSCS working process with sufficient adequacy can be described by the mathematical tools of queuing theory (Kleinrock, 1979). Thus, according to the classical works (Khinchin, 1963), various experimental data queries to a database can be described as a Poisson events flow: according to the Poisson distribution, the number of events k in the time interval t is distributed with the following probability:

$$P(k,t) = \frac{(\lambda t)^k e^{-\lambda t}}{k!}, \tag{1}$$

where λ is the intensity of the events flow. In this case, the superposition of Poisson flows is a Poisson flow with total intensity.

Further, for the 'typical' query service time can be represented by a discrete random variable, i.e. the probability of service time bj of a *j*-type query:

$$P(b_j) = \frac{\lambda_j}{\sum_k \lambda_k}, \tag{2}$$

where $\lambda_j (\lambda_k)$ is the intensity of the flow of the *j*(*k*)-type queries.

Considering this model, it is possible to use the average waiting time of the query as the basis of the efficiency criterion. For one database the efficiency criterion for processing the group of data classes can be variable:

$$w = \frac{\lambda \sigma_B^2}{2(1-\lambda b)}, \tag{3}$$

where *b* is query service time expectation for this database in accordance with its load,

σ_B^2 - query service time variance.

In this paper we do not consider the interaction of DBMSs and data consistency. The relationship between classes could be represented as a sequence of queries.

To use of the average query service time for a system of multiple databases is obviously wrong. As the efficiency

criterion of the total system of multiple databases (i.e., the criterion of DSCS efficiency) the following value is proposed:

$$J = \sum_i \sum_j (b_{ij} + w_i) x_{ij} , \tag{4}$$

where w_i is an i-type database efficiency criterion,

b_{ij} – j-type query service time in i-type database (let us denote $j \to i$),

x_{ij} – belonging to set element:

$$x_{ij} = \begin{cases} 1, j \to i \\ 0, \overline{j \to i} \end{cases}. \tag{5}$$

The value of the criterion of DSCS efficiency for a fixed set of data classes and used software solutions is, in fact, the evaluation of the efficiency of data separation to the different databases.

Next, to determine the optimal, in terms of the proposed criteria, way of dividing the stored data, it is necessary to solve the problem of 70ptimizati the variables x_{ij}, where criterion J is the objective function and the following are the restrictions:

$$\begin{cases} 1 - \lambda_1 b_1 > 0 \\ \ldots \\ 1 - \lambda_M b_M > 0 \end{cases}, \tag{6}$$

where λ_i is the total intensity of the query flow for the i-type database,

b_i – the expectation of query service time for the i-type database,

M – number of databases.

Submitted constraints determine the absence of a queue accumulation for each database. Any linear constraints can be added further to define, for example, a strict affiliation of the respective classes to a database.

The problem of 70ptimization criterion J belongs to a class of nonlinear pseudo-Boolean problems (Boros, Hammer, 2001). For a practical problem of 100 data classes and 2 databases there are approximately 1030 possible 70ptimization. Pseudo-Boolean 70ptimization problems are often encountered in practice; there are many approaches to solve them. However, in view of large dimensions and high computational complexity, there is no universal algorithm that allows an exact solution in a reasonable time.

3. Optimisation of the Data Partitioning

To solve the problem we must find a proper partitioning

$$X^* : J(X^*) = \min_J (J(X)), X \in \mathrm{X} , \tag{7}$$

where X is a data partition,

X – all possible partitions.

The criterion J is a nonlinear function of variables x_{ij}, which brings difficulty in future 70inearizatio. For a formal description of the problem of pseudo-programming, we introduce the following notation:

$S \subseteq \{1,\ldots,N\}, N \in \mathrm{N}$ - the set of integers with characteristic vector:

$$1^S \in \{0,1\}^N : 1_j^S = \begin{cases} 1, j \in S \\ 0, j \notin S \end{cases}, \tag{8}$$

$\mathrm{u} = \{u_1,\ldots,u_N\}, u_j \in \{0,1\}$ - binary (Boolean) vector of N variables,

$f(\mathrm{u}) : \{0,1\}^N \to \mathrm{R}$ - pseudo-Boolean function of N variables.

According to (Boros, Hammer, 2001), all pseudo-Boolean functions could be uniquely represented as multi-linear polynomials:

$$f(u_1,\ldots,u_N) = \sum_{S \subseteq \{1,\ldots,N\}} c_S \prod_{j \in S} u_j . \tag{9}$$

The size of the largest subset S: cS≠0 is called a degree of f and is denoted by deg(f). We will call a pseudo-Boolean function linear, if *deg(f)*≤1, and linear-fractional, if it could be represented as a fraction of two linear functions:

$$f(\mathrm{u}) = f_1(\mathrm{u}) / f_2(\mathrm{u}), \begin{cases} \deg(f_1) \le 1 \\ \deg(f_2) \le 1 \end{cases}. \tag{10}$$

In general, it is difficult to transform a function into a multi-linear form due to a large number of additional variables and constraints.

The integer programming problems and, in particular, problems of pseudo-Boolean 71 inearizatio are well-known. There are many approaches to solve them, but the lack of versatility and productivity of the algorithms makes it necessary to find the proper algorithm for solving each particular problem.

In general, the algorithms can be classified in terms of accuracy of the solution (exact and approximate) and belonging to a certain class of functions (linear, linear-fractional quadratic problem, and others.).

In some cases, it is appropriate to use the branch and bound algorithm, the effectiveness of which is determined by the accuracy of estimates of the upper and lower limits of the values of the function being 71inearіza (Martello, Toth, 1990).

Given the nature of the criterion proposed in the paper, as well as reasonable restrictions on the number of classes of data, we propose to exercise partial 71inearization of criterion *J*, resulting in the linear-fractional function with the introduction of additional Boolean variables and constraints. The obvious disadvantage of this approach is an increasing of the number of variables, according to dimension of the criterion *J*.

3.1 Partial linearisation

Linearisation is a standard technique to reduce nonlinear binary optimization to linear integer programming. The basic idea is to replace a nonlinear term with a new variable subject to additional constraints and so force it to take in all feasible solutions of the value of the original term. For instance, we could replace a product of two binary variables $x, y \in \{0,1\}$ by new binary variable *u* with constraints:

$$\begin{cases} u \le x \\ u \le y \\ u \ge x + y - 1 \\ u \ge 0 \end{cases}, \tag{11}$$

With these constraints, values of the variable u are identical to corresponding values of the product xy:

Table 1. Replacement of the product of binary variables

x	y	xy=x&y	u
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

Similarly, in the case of replacing product of L values x1, …, xL, we will add L+2 constraints of the following form:

$$\begin{cases} u \le x_1 \\ \dots \\ u \le x_L \\ u \ge x_1 + \dots + x_L - (L-1) \\ u \ge 0 \end{cases}. \tag{12}$$

During the optimization, seeing $x_{ij} \in \{0,1\}$, it is obvious that:

$$\forall i, j \Rightarrow x_{ij}^2 = x_{ij}; \forall i_1, j_1, i_2, j_2 \Rightarrow x_{i_1 j_1}^{k_1} x_{i_2 j_2}^{k_2} = x_{i_1 j_1} x_{i_2 j_2}. \tag{13}$$

Let us denote:

$\boldsymbol{u}=\{u_1,\ldots,u_H\}$ – a vector of u variables that reduces criterion J (4) to linear-fractional form (10), and H is a dimension of the vector.

Clearly, $\mathrm{u} \in \{1,0\}^H$, i.e. the problem is still pseudo-Boolean.

Denoting

$$w_i=w_{1i}/w_{2i}, \tag{14}$$

$$\text{it could be seen that } \begin{cases}\deg(w_{1i}) \le 2\\ \deg(w_{2i}) \le 2\end{cases}, \tag{15}$$

i.e., w_i is a fraction of two quadratic functions. Furthermore, for notational convenience, we will assume that *deg(f)=k* if *sup(deg(f))=k*. With this notation (omitting index *i*):

$$w = \frac{w1}{w2} = \frac{\alpha_0 + \sum_{j_1,j_2=\overline{1,N},j_1\le j_2} \alpha_{j_1j_2} x_{j_1} x_{j_2}}{\beta_0 + \sum_{j_1,j_2=\overline{1,N},j_1\le j_2} \beta_{j_1j_2} x_{j_1} x_{j_2}}, \tag{16}$$

we will have:

$$\begin{cases}\alpha_0 = 0\\ \alpha_{j_1j_2} = 0, j_1 = j_2\\ \alpha_{j_1j_2} = \lambda_{j_1}\lambda_{j_2}(b_{j_1} - b_{j_1})^2, j_1 < j_2\end{cases}, \tag{17}$$

$$\begin{cases}\beta_0 = 0\\ \beta_{j_1j_2} = \beta_j = 2\lambda_j(1-\lambda_j b_j), j_1 = j_2 = j\\ \beta_{j_1j_2} = -2\lambda_{j_1}\lambda_{j2}(b_{j_1} - b_{j_2})^2, j_1 < j_2\end{cases}. \tag{18}$$

Furthermore, by observing that *j*-type query 'real' service time (service time with waiting time) in *i*-type database:

$$\tilde{b}_{ij} = \tilde{b}_{1ij} / \tilde{b}_{2ij}, \tag{19}$$

we obtain:

$$\begin{cases}\deg(\tilde{b}_{1ij}) = 3\\ \deg(\tilde{b}_{2ij}) = \deg(w_{2i}) = 2\end{cases}. \tag{20}$$

Given by (14) the independence w_{2i} of j, we could represent:

$$J = \sum_i 1/w_{2i} \sum_j \tilde{b}_{1ij} = J_1 / J_2, \tag{21}$$

and hence we have:

$$\begin{cases}\deg(J_1) = 2M+1\\ \deg(J_2) = 2M\end{cases}. \tag{22}$$

Considering the optimization72, for the pseudo-Boolean function $L(x_1,\ldots x_{MN})$: $deg(L)=l$ the amount of additional variables could be limited by:

$$H = H_v \le \sum_{t=2}^{l} C_{MN}^t, \tag{23}$$

and the amount of additional constraints would be no more than:

$$H_c \le \sum_{t=2}^{l} (t+2) C_{MN}^{t} . \tag{24}$$

According to replacement (12) we could reduce the problem of optimization of the optimizat criterion *J(X)* to the pseudo-Boolean linear-fractional optimization problem.

It is necessary to consider the case M=2, in which: $\forall j = \overline{1,N} \Rightarrow x_{1j} = \overline{x_{2j}}$, that greatly simplifies the final expression of *J(X)* and, in fact, reduces the number of variables $x_{11}, \ldots, x_{MN}$ by half.

In this case, given x_{2j}=1-x_{1j}, and with notation:

$$\begin{cases} w_1 = w \\ w_2 = \overline{w} \end{cases}, \tag{25}$$

$$\begin{cases} x_{1j} = x_j \\ x_{2j} = \overline{x_j} = 1 - x_j \end{cases}, \tag{26}$$

expressions (16), (17), (18) could be similarly simplified, also we would be able to replace MN by N in expressions (23), (24), that greatly simplifies the further optimization.

3.2 Approaches to Solve the Linear-Fractional Problem

After the partial linearisation, the criterion J could be represented:

$$J_L = F_{1L}(u_1, \ldots, u_H) / F_{2L}(u_1, \ldots, u_H),\ deg(F_{1L}) = deg(F_{2L}) = 1,$$

Hence, let us denote:

$$J_L = \frac{c_0 + c_1 u_1 + \ldots + c_H u_H}{d_0 + d_1 u_1 + \ldots + d_H u_H} . \tag{27}$$

In fact, $J_L = J_L(x_{11}, \ldots, x_{MN}, u_{MN+1}, \ldots, u_{H\text{-}MN})$, i.e., we replace only products, but, to simplify the notation we will replace (rename) all x-variables in such a way:

$$\begin{cases} u_j = x_{kl}, k = \overline{1,M}, l = \overline{1,N}, j = N(k-1) + l \\ u_j = \prod_S x_{kl}, j = \overline{MN, H} \end{cases} . \tag{28}$$

It should be noted that the solution of the optimisation problem may be the only $\boldsymbol{u}$, for which: $F_{2L}(\boldsymbol{u})$>0.

That follows from the initial constraints of the form:

$$\forall i = \overline{1,M} \Rightarrow \sum_j (\lambda_j b_{ij} x_{ij}) < 1 . \tag{29}$$

Linear-fractional pseudo-Boolean problems often occur, for example, in clustering, when we have to decide, if an element belongs to some cluster or not (Prokopyev, 2006).

Furthermore, if the denominator F_{2L} could take both negative and positive values, the optimisation problem is *NP*-hard. It could be shown, that optimising can't be easier than finding a solution to the subset sum problem, a well-known *NP*-complete decision problem.

3.2.1 Complete Linearisation with Additional Non-Integer Variables

In general case, it is possible to reduce the linear-fractional problem to linear form:

we could replace the denominator by a new variable:

$$z = \frac{1}{d_0 + d_1 u_1 + \ldots + d_H u_H} \tag{30}$$

and, denoting $v_h = u_h z$, we will have new linear problem with variables: $v_1, \ldots, v_H, z$:

$$J_{fL}(v_1, \ldots, v_h, z) = c_0 z + \sum_{j=1}^{H} c_j v_j , \tag{31}$$

with additional constraint: $d_0z+d_1v_1+...+d_Hv_H=1$. Due to non-integer variable z the problem transforms into non-integer linear optimisation problem. After solving this problem we have to make further iterations (for example, cutting-plane method) to find a integer 0-1 solution $u_1,..,u_H$.

3.2.2 Heuristic Approaches Based on the GRASP Algorithm

As one of the approaches to solve the optimisation problem we could consider heuristic algorithms that are capable of providing a proper solution with a reasonable amount of time and sufficient accuracy (Brady, Catanzaro, 2008). As an example, let us consider a GRASP (Greedy Randomised Adaptive Search Procedure) – an approach that is widely used for the construction of a heuristic algorithm for optimisation problems (Mauricio, Resende, 1998). A GRASP typically consists of iterations made up from the successive construction of a greedy randomised solution and subsequent iterative improvements of it through a local search.

This paper proposes the use of this approach only if the dimension of the problem does not allow an exact solution in a reasonable time. We take into account that the problem of data partitioning is the task of planning, i.e., the solution of the problem in real-time is not required and the solution accuracy is more important than time.

3.2.3 Reducing the Optimisation Problem to the *SAT*-Problem

One of the approaches to the optimisation problem is to reduce the issue to a fairly well-known Boolean Satisfiability Problem (*SAT*-problem) (Een, Sorensson, 2006). *SAT*-problem is a *NP*-complete problem; there are many algorithms which solve it with different efficiency.

The essence of the transformations considered in the optimisation problem is the following (Crama, Hansen, Jaumard, 1990):

1. Preparation of a solution satisfying the constraints (the construction a workable solution);
2. Adding a new constraint to cut off the solution found.

There are many programs solving SAT-problems (SAT-solvers), most of which deal with linear constraints.

The general scheme for solving the problem is as follows:

The partially linearised problem could be represented:

$$\begin{cases} J_L(u_1,...,u_H)=\dfrac{c_0+c_1u_1+...+c_Hu_H}{d_0+d_1u_1+...+d_Hu_H}\to\min \\ L_1(\lambda_j,b_{ij},x_{ij}) \\ L_2(x_{ij},u_{MN+1},...,u_H) \end{cases}, \tag{32}$$

where $L_1(\lambda_j,b_{ij},x_{ij})$ - M constraints by form (29),

$L_2(x_{ij},u_{MN+1},...,u_H)$ - H_c constraints with additional variables by form (12).

The solution of the *SAT*-problem will be the vector $\boldsymbol{u}=\boldsymbol{u}^*$ (the set values $(x_{11},\ldots,x_{MN},u_{MN+1},\ldots,u_{H\text{-}MN})$ or given renaming, $u_1,\ldots,u_H$) which satisfy the constraints L_1 and L_2. Let us denote:

$$\mathrm{u}^*=(u_1^*,...,u_H^*). \tag{33}$$

Furthermore, we will add constraint by form:

$$J_L(\mathrm{u})<J_L^*=J_L(\mathrm{u}^*) \tag{34}$$

and when we solve an extended *SAT*-problem, we will have another solution $\boldsymbol{u}^{**}$: $J_L(\boldsymbol{u}^{**})<J_L(\boldsymbol{u}^*)$. Denoting $\boldsymbol{u}^*=\boldsymbol{u}^{**}$, we will have an iteration cycle, at each iteration the solution will improve the value of the objective function. For the global minimum we should continue as long as the extended *SAT*-problem is solvable. The solution that we have will be an optimum, hence the data partitioning

$$X^*=(x_{11}^*,...,x_{MN}^*) \tag{35}$$

will be the optimal required partitioning.

It is necessary to consider that additional constraints are linear and could be represented by form:

$$L_a = (c_0 - J_L^* d_0) + \sum_{j=1}^{H} (c_j - J_L^* d_j) u_j < 0 . \quad (36)$$

Also, in each iteration we will have the same *SAT*-problem with one changing constraint.

In the base of *SAT*-solvers there is a backtracking based search algorithm *DPLL* (Davis-Putnam-Logemann-Loveland). Despite the high theoretical complexity of the algorithm, program *SAT*-solvers are sufficiently effective in practice (Hossein, Sheini, Karem, Sakallah, 2006). According to results of yearly competitions, modern *SAT*-solvers allow the solution of problems within a reasonable time, even when they consist of about 10^6 variables and $3 \cdot 10^6$ constraints (Jarvisalo, Le Berre, Roussel, 2013).

3.2.4 Simplified Optimisation with Strict Constraints

Considering the case in which, for all u value $F_{2L}(\boldsymbol{u})>0$, we can see that the solution could be found as follows:

Without loss of generality let us assume: $d_h>0$, $h=0,\ldots,H$; $c_h>0$, $h=1,\ldots,H$:

- simultaneously $c_{h0}<0$ and $d_{h0}<0$, let us denote: $u_{h0}=1-u_{h0}$;
- $c_{h0}<0$, $d_{h0}>0$, let us define $u_{h0}=1$;
- $c_{h0}>0$, $d_{h0}<0$, similarly, we define $u_{h0}=0$;

Furthermore, let us denote $\boldsymbol{u}^*$ - desired vector (optimum of J), then, if we assume $J(u^*) \le t$, we will have:

$$(c_0 - td_0) + \sum_h (c_j - td_j) u_h^* \le 0 , \quad (37)$$

or

$$\max_u \sum_h (c_j - td_j) u_h^* \le -c_0 + td_0 . \quad (38)$$

If we reorder coefficients in such a way that:

$$\frac{c_{h1}}{d_{h1}} \le \frac{c_{h2}}{d_{h2}} \le \ldots \le \frac{c_{hH}}{d_{hH}} , \quad (39)$$

we will have $\boldsymbol{u}^*$ as one of the H vectors of the form:

$$\begin{cases} u_{hl} = 1, l \le k \\ u_{hl} = 0, l > k \end{cases}, k = \overline{0, H} . \quad (40)$$

If all the vectors of (40) satisfy the constraints L_1 and L_2 (32), then, obviously, the solution could be found faster than with iterative methods. The computational complexity could be estimated as:

$$\max(O(H \log H), O(H_c)) , \quad (41)$$

because reordering the coefficients requires $O(H \log H)$ operations, searching and comparing of corresponding values of the function $J - O(H)$ operations, additional constraint verification - $O(H_c)$.

Given the limits of the practical problem (2 databases, less than 100 data classes), we propose:

- To make an attempt to solve the simplified optimisation problem;
- In the case of the non-constant sign of $F_{2L}(\boldsymbol{u})$, to reduce the pseudo-Boolean problem to *SAT*-problem and to solve it with one of the open-source *SAT*-solvers.

As an example, let us consider a partition of 10 classes into 2 databases; the input data are the intensity and the time of query processing:

Table 2. The input data

j	1	2	3	4	5	6	7	8	9	10
λ_i ,s^{-1}	0.062	0.125	0.25	0.5	1	0.2	0.1	0.1	0.15	0.1
b_{1j}, s	5.5	1.5	1.2	0.18	0.125	0.3	0.8	0.1	0.1	0.21
b_{2j}, s	2.0	1.0	0.8	0.45	0.225	0.1	0.3	0.1	0.2	0.31

For the initial approximation $X_0=\{x_{ij}\}$:

Table 3. The initial approximation

*i**j*	1	2	3	4	5	6	7	8	9	10
1	0	0	1	0	1	1	0	1	1	0
2	1	1	0	1	0	0	1	0	0	1

Value of the criterion J=12.8s, the average waiting time of execution of applications is 0.69s (i=1), 0.66s (i=2).

Optimal partition X_{opt} is the following:

Table 4. The optimal partition

*i**j*	1	2	3	4	5	6	7	8	9	10
1	0	0	0	1	1	1	0	1	1	1
2	1	1	1	0	0	0	1	0	0	0

Value of the criterion J=6.07s, the average waiting time for the execution of applications is 0.01s (i=1), 0.22s (i=2).

The choice of software solutions as the basis for the ACS design is caused, first of all, by characteristics of the subject area and the stored data. To solve the practical problems it is most useful to combine software solutions with different architectures for the 'widest' coverage of processed data characteristics. However, as a kind of limiting case, the described technique can be applied to the equal databases. In such cases, this configuration is a highly scalable system with a permanent partitioning of the data classes. Also, the criterion for evaluating the effectiveness of a single database can be used when choosing the right software solutions regardless of data partitioning.

4. Conclusion

Thus the article describes the technique of evaluating and improving data storage control systems' efficiency. This technique is based on a mathematical model. Despite limitations in equipment capabilities and communication channels, it enables all functions and effectively completes the task of different-type data storage during ACS design.

References

Boros, E., & Hammer, P. L. (2001). *Pseudo-Boolean Optimization*. RUTCOR. http://dx.doi.org/10.1016/S0166-218X(01)00341-9

Brady, B., & Catanzaro, B. (2008). *Pseudo-Boolean Heuristics for 0-1 Integer Linear Programming*. Berkeley.

Carey, M. J., & Livny, M. (1988). *Distributed Concurrency Control Performance: A Study of Algorithms, Distribution, and Replication.* University of Wisconsin, Computer Sciences Department.

Crama, Y., Hansen, P., & Jaumard, B. (1990). *The basic algorithm for pseudo-Boolean programming*. RUTCOR.

Documentation on 'Mirror Database Tools' means of increasing the performance of databases. (2014). Retrieved from http://www.mirrordatabase.com

Documentation on distributed system Apache Hadoop. (2014). Retrieved from http://www.hadoop.apache.org

Documentation on distributed system The Google File System. (2014). Retrieved from http://www.research.google.com

Documentation on memcached storage system. (2014) Retrieved from http://www.memcached.org

Documentation on Oracle database. (2014). Retrieved from http://www.docs.oracle.co

Documentation on PostgreSQL database. (2014). Retrieved from http://www.postgresql.org

Documentation on SQLite database. (2014). Retrieved from http://www.sqlite.org

Dudakov, N. S., Pirogov, N. E., & Shumilov, Y. (2009). *Cross-platform management system for secure storage of dynamic data*. SIT #3. M.: NRNU MEPhI.

Een, N., & Sorensson, N. (2006). Translating Pseudo-Boolean Constraints into SAT. *Journal on Satisfability,* Boolean Modeling and Computation. Berkeley.

Ensor, D., & Stevenson, I. (1999). '*Oracle. Database Design*'. K.: Publishing Group BHV.

Gilbert, Seth, & Lynch, N. (2012). *Perspectives on the CAP Theorem*. MIT, *Computer, 45*(2). http://dx.doi.org/10.1109/MC.2011.389

Hossein, M., Sheini, K., & Sakallah, A. (2006). *Pueblo: A Hybrid Pseudo-Boolean SAT Solver*. Journal on Satisfiability, Boolean Modeling and Computation.

Jarvisalo, M., Berre, L., & Roussel, D. O. (2013). *The International SAT Solver Competitions*. Retrieved from www.satcompetition.org. http://dx.doi.org/10.1609/aimag.v33i1.2395

Khinchin, A. Y. (1963). *Work on mathematical queuing theory*. Moscow.

Kleinrock, L. (1979). *Queueing Theory*. M.: Engineering.

Martello, S., & Toth, P. (1990). *Knapsack problems*. Wiley.

Mauricio, G. C., & Resende. (1998) *Greedy Randomized Adaptive Search Procedures (GRASP)*. AT&T Labs Research.

Prokopyev, O. A. (2006). *Nonlinear integer optimization and applications in biomedicine*. University of Florida.

Wiegers, K. (2004). *Developing software requirements*. Moscow: Publishing and Trading House 'Russian edition'.