

The column-oriented database partitioning optimization based on the natural computing algorithms

Artur Nowosielski¹

¹PhD Studies, Systems Research Institute
Polish Academy of Sciences
ul. Newelska 6, 01-447 Warsaw, Poland
Email: artnowo@gmail.com

Piotr A. Kowalski^{2,3}, Piotr Kulczycki^{2,3}

²Systems Research Institute
Polish Academy of Sciences
ul. Newelska 6, 01-447 Warsaw, Poland
Email: {pakowal,kulczycki}@ibspan.waw.pl

³Division for Information Technology and Biometrics
Faculty of Physics and Applied Computer Science
AGH University of Science and Technology
al. Mickiewicza 30, 30-059 Cracow, Poland
Email: {pkowal,kulczycki}@agh.edu.pl

Abstract—This paper describes the basic components of a research project aimed at the application of natural computing metaheuristics to optimize the horizontal scaling of databases. Column oriented databases were selected for the project because of their unique properties. A mathematical model has been created in order to align the problem of horizontal scalability to the general optimization methods, such as natural computing algorithms.

I. INTRODUCTION

THIS article is an overview of research on column oriented databases (DB) partitioning optimization with the use of the natural computing algorithms. Column-oriented databases are believed to qualify for this purpose nicely thanks to their physical storage structure. Sometimes they are used as a NoSQL equivalent for relational database management systems because of their flexibility and partial similarity to relational model. In the class of the natural computation algorithms there are metaheuristic procedures that suite well to the problems of optimization with multiple constraints and multi-modal objective functions.

There are three main pillars of the research, described in the following subsections. Subsection I-B contains a general description of the column-oriented databases along with the most important relevant details. From the scalability options for databases, horizontal scalability gained some noticeable attention and major implementations, especially in a modern so-called “web 2.0” services. Subsection I-A describes basic features along with pros and cons of a database horizontal scaling. The natural computing algorithms described in the section I-C are an important branch of the research on computation models and methods. Their main goal is to implement heuristics inspired by the natural environment’s behaviors and processes, to solve optimization problems.

The next sections describe current state and results of

the aforementioned research. They consists of the prototype implementation of a column-oriented DBMS along with its mathematical model. Selected natural computing algorithms were also implemented in order to discover and describe their features on the basis of the typical benchmarking optimization problems. Section III contains the description of the application of the algorithms to the optimization problem using a created mathematical model. Current short- and long-term plans for further research are described in order to familiarize the reader with the expectations of the research.

A. Database partitioning

In general, two main classes of scalability solutions can be distinguished: vertical scaling and horizontal scaling. These approaches are slightly different. Vertical scalability is based on the assumption that, in order to increase system capacity, its resources should be enforced, e.g. by swapping the CPU to a faster one. In this approach, a unit of work remains assigned to only one processing unit and is therefore limited by the unit’s capabilities. Additionally, hardware capabilities are finite and it is possible to reach to the point in which it will not be possible or reasonable to deploy a more powerful hardware. Instead, this approach is easy to implement, because it preserves previously used computation model and processing methods. Especially, it does not involve any work division, which means no additional synchronisation issues are introduced. In the horizontal scalability (scaling out) approach, it is assumed the unit of work to be done exceeds the capacity of any single unit and needs to be divided. That slightly changes the processing model and introduces concurrency- and synchronisation-related issues, but in return, offers virtually infinite theoretical capacity.

Within the database domain, horizontal scalability can be split into the three classes: replication, sharding (also referred

to as a horizontal partitioning) and partitioning (also known as the vertical partitioning). Please note that the terms “vertical” and “horizontal” regarding partitioning and sharding are absolutely not related to the general scalability terms. Both are parts of the horizontal scalability class. That is, both involve some division of a data in a database into more than one database management system (DBMS) instance. Replication is out of the research scope, so it will not be described here. It has been covered widely, along with the remaining solutions in the master thesis [1].

Horizontal and vertical partitioning differ by a division plane. Horizontal partitioning duplicates database (or just single table) schema on many instances but splits the data between them. Please note that specific partitions can be distinctive (that is, every records belongs to one and only instance) as well as redundant. Partitioning function can take into consideration record’s identifier, single other field or a set of fields. A value which is used to perform partitioning is so-called “partitioning key”. Typical partitioning schemas include:

- by range (e.g. clients with last name starting with letters A, B, ..., I, J go to the instance#1, K, L, ..., Y, Z - #2);
- by modulo function (e.g. if record have natural numeric key they go to instance number (key % n) where n = number of instances);
- by list (if record has a field constrained by a finite set of values, instances can be assigned with its subsets; e.g. clients born on monday, tuesday and friday go to the first instance, others to the second one);
- by hash function (hash function result set must reflect instance set).

More sophisticated methods include combinations of previously enumerated ones and consistent hashing. Consistent hashing is a technique which optimizes a division for frequent remapping, reducing number of data to be migrated after every change.

Very useful concept here is the abstraction of hash function results from the physical arrangement of database engine instances. That requires additional mapping between these two tiers, but brings some significant advantages of a direct linkage. Primarily, changes in the arrangement of database servers does not involve modifications of the hash function. In other words, it satisfies well-known “single responsibility principle” coming from the software engineering domain.

Sharding can then be considered as a data division between duplicated schema instances. Vertical partitioning, in turn, can be thought as a schema division. It requires a deep analysis of the data usage and implicitly assumes that structure of every records is common if not exactly the same. Then rarely used parts of data are extracted and moved to a different instance(s) than the frequently used parts. That is not the only possible analysis schema though. It is also possible to highlight data parts which are frequently used together and divide a database by that. Obviously, in real world application both techniques can be joined and applied together in any arrangement, which leads to virtually infinite number of possible variants.

B. Column-oriented databases

Column-oriented databases (sometimes referred to as a column-family databases, columnar databases or column stores) is a non-relational database data model. Although not strictly defined, this model brings some significant features increasing its horizontal scalability. The current section describes the model as it is understood within the conducted research. For the ease of understanding, analogies to the well known relational database model are highlighted. Nevertheless, it needs to be stated explicitly that both models are not related. Modern column-oriented data stores have been covered in [2]. Implementation details of the Apache Cassandra column-family store have been described in [3].

Fundamental terms and concepts are common to every column-oriented DBMS implementation. Keyspace is the basic storage and logical entity. It acts as a container for lower level entities and could be compared to database or schema in terms of relational databases. A column family is a named collection of records with similar or the same column set. Column families belong to the keyspaces.

The most important feature coming from the described concepts is so-called schemalessness, which means that column family’s data structure is flexible. A column set for specific records within the same column family is variable. In real world solutions it is usually “defined” only by an established convention, not by a strict constraints stated during the database creation process. As opposed to the relational database tables, if a given record does not have (in a logical sense) value in given column, it does not store actual NULL value (in a storage sense). That feature brings a significant impact on the storage structure and application logic. Also, it allows to implement some use cases which are not effective or even possible in relational databases. A common use case is the time-series data store, which contains actually only one row of data, but with dozens of columns created every time a new value is saved. In that case, the value timestamp becomes a column name. Another typical appliance is a data store for a recommendation system. Such systems rely on huge matrices correlating all users with all items.

C. Natural computing algorithms

Natural computing algorithms are a significant class of the heuristic procedures, which takes its inspiration in processes and behaviors taken from nature. According to [4], natural computing algorithms can be divided into three main categories:

- evolutionary algorithms;
- swarm intelligence algorithms;
- bacterial foraging algorithms.

They are all based on an observation, that many processes in nature are in fact non-linear, multi-modal and multi-objective optimization processes. The whole evolution process leads to survival of the fittest specimens and species. Within that process there are many constraints, most of which are not well-defined or even not well-known, in some cases they could be

fuzzy. Obviously, individuals do not use any numeric, analytical methods to fit, because it is not possible. That conclusion leads to arise of metaheuristic algorithms, which follow the nature patterns and can be used to perform optimization under similar conditions. For the sake of correct understanding the algorithm's abstraction, it is very important to emphasize analogies between the biological domain and optimization domain. Evolution's goal (in nature it is simply survival and reproduction of given individual's genes) is reflected by the objective function, which is minimized or maximized during the computation. As the algorithms are iterative, iteration loop reflects the lapse of time and generations succession. Candidate solutions are equivalents of the real population members and they pass from generation to generation using rules which differ between the algorithms and applications. In contrast to nature, algorithms are initialized (that is populated with an initial population) with random solutions to provide some reasonable starting point for the algorithm execution. It is very important to note that natural computation algorithms, as they are metaheuristics, do not guarantee finding optimal solution. Although, given enough population size and generation number, they usually lead to the suboptimal solutions which are suitable enough for the most of applications.

Two algorithms for the evaluation and application were chosen in the described research: the Flower Pollination Algorithm (FPA) and the Krill Herd Algorithm (KHA).

1) *Flower Pollination Algorithm*: The first one has been proposed by Xin-She Yang of Department of Engineering, University of Cambridge in the paper [5] with further description in the paper [6]. It belongs to the class of evolutionary algorithms and it reflects the process of reproduction of flowering plants by the pollination. It is performed by pollinators, such as insects, birds or bats (biotic pollination) or by the wind or water (abiotic pollination). Pollination can take the form of a cross-pollination, in which a flower is pollinated with pollens coming from flowers of different plant, or a self-pollination, in which flowers on the same plant pollinate each other. Some pollinators stick to some species and specialize in pollinating them. This phenomenon is called flower constancy. Author highlighted four abstractions which make up the basis of the FPA:

- global pollination is an abstraction of biotic cross-pollination, pollinators move by performing Lévy flights;
- local pollination is an abstraction of abiotic self-pollination;
- flower constancy is expressed by the fact that reproduction probability increases appropriately to similarity of the two involved flowers;
- probability of occurrence of local and global pollination is defined by the switch probability p , which abstracts external factors, such as wind, physical proximity of different plants and so on.

There are some additional assumptions and rules as well:

- the best solution from the generation g_i passes directly to the next generation g_{i+1} ;

Algorithm 1 FPA pseudocode

```

Objective min or max  $f(X)$ ,  $X = (X_1, X_2, \dots, X_d)$ 
Initialize a population of  $n$  flowers/pollen gametes with
random values
Find the best solution  $g_*$  in the initial population
Define a switch probability  $p \in [0, 1]$ 
while  $t < MaxGeneration$  do
  for  $i = 1..n$  do
    if  $rand < p$  then
      Draw a ( $d$ -dimensional) step vector  $L$  which obeys
      a Lévy distribution
      Global pollination via  $X_i^{t+1} = X_i^t + L(X_i^t - g_*)$ 
    else
      Draw  $\epsilon$  from an uniform distribution in  $[0, 1]$ 
      Randomly choose  $j$  and  $k$  among all the solutions
      Local pollination via  $X_i^{t+1} = X_i^t + \epsilon(X_j^t - X_k^t)$ 
    end if
    Evaluate new solution
    If the new solution is better, update it in population
  end for
  Find the current best solution  $g_*$ 
end while

```

- with global pollination, candidate solutions tend to the current most fit individual;
- a random value determining pollination type is obtained for every flower in every iteration;
- newly generated solution passes to the next generation if, and only if, it is better fitted than its predecessor.

Assuming that the current best solution is represented by g_* , generating solution x_i in step $t + 1$ in global pollination is performed by the following formula:

$$X_i^{t+1} = X_i^t + L(X_i^t - g_*) \quad (1)$$

where $L > 0$ is acquired from the Lévy distribution. A local pollination version is in turn expressed by the formula:

$$X_i^{t+1} = X_i^t + \epsilon(X_j^t - X_k^t) \quad (2)$$

where ϵ is a value obtained from the regular uniform distribution.

The FPA has been benchmarked with some typical two dimensional functions enumerated in the original FPA article, as well as custom function created especially for the sake of research. Tests were performed on the implementation created in the SciPy environment using a small custom test framework. Tunable parameters of the algorithm were changed between specific test runs, which led to the following conclusions:

- finding a solution for the unimodal problem requires much less effort (in terms of generation number and population size) than for the multimodal problem;
- probability switch parameter p does not matter significantly in unimodal problems;
- given enough iterations, every other parameter also does not really matter in unimodal problem solving;

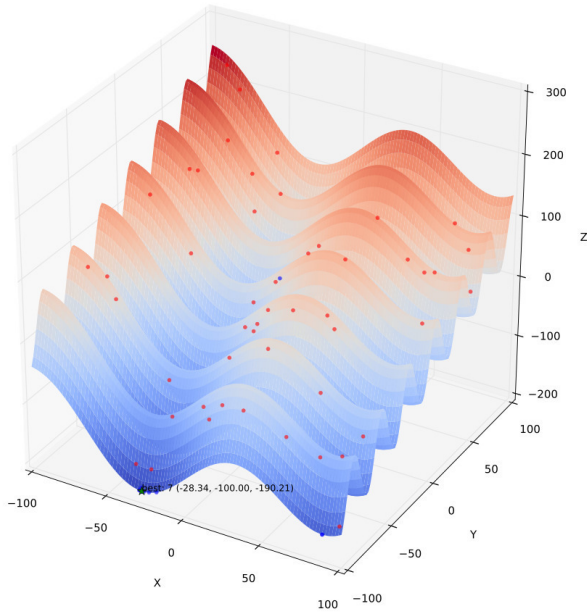


Fig. 1. Results of the application of FPA to the custom benchmark two-dimensional function.

- for multimodal problems, increasing population size gave significantly less improvement than increasing iteration number;
- in extreme cases, during the few first generations there was no improvement in comparison to the initial random population (!);
- high values of the pollination switch parameter p decreased effectiveness by sticking the computation to local minimum values in a multimodal problems.

Figure 1 presents a plot of the custom two dimensional benchmark function. Its objective function is the minimum in the domain of $[-100, 100]$ in both dimensions.

$$f(x) = \left(\left(\frac{x_i}{20} \right)^2 - 2 \right) \left(\frac{x_i}{20} + 2 \right) - 2 + 50 \sin \left(\frac{x_i}{40} \right) + 10 \sin \left(\frac{\left(\frac{x_i}{20} \right)^2}{2} \right) \quad (3)$$

Red dots are individuals of the initial flower population, while blue dots are individuals of the population in the last generation. Green stars mark the most fit flower in the last generation along with its coordinates.

2) *Krill Herd Algorithm*: The Krill Herd Algorithm represents a slightly different category of metaheuristic nature-inspired algorithms from the FPA. It is a swarm intelligence algorithm. Originally proposed by Amir Hossein Gandomi and Amir Hossein Alavi in the paper [4], it mimics the behaviour of the individual krill specimens moving together as a herd. Such herds, or swarms, move accordingly to environmental factors, but every krill moves separately. An individual's movement is determined by three factors:

- the movement vector of the whole swarm (neighbours within the swarm);
- food foraging;
- additional random bias (diffusion).

After removal of an individual krill (caused by predator attack) from a herd, krills tend to "fix" the low-density gap while still being oriented on finding food. From this emerges a multiobjective optimization problem. Overall, generalized n -dimensional formula for difference of krill position in subsequent time units goes as follows:

$$\frac{dX_i}{dt} = N_i + F_i + D_i \quad (4)$$

Enumerated aspects of individual krill moves can be described by a set of equations:

- N_i - motion induced by neighbours:

$$N_i^{new} = N^{max} \alpha_i + \omega_n N_i^{old} \quad (5)$$

where N^{max} is the maximum possible speed that can be induced, ω_n in the range $[0, 1]$ is the inertia weight of an individual krill. N_i^{old} is the motion induced in the previous turn and

$$\alpha_i = \alpha_i^{local} + \alpha_i^{target} \quad (6)$$

α_i^{local} is the local influence of the neighbours on an individual krill, while α_i^{target} is the target direction. Target is determined by the position and movement of the best individual in a swarm.

$$\alpha_i^{local} = \sum_{j=1}^{NN} \hat{K}_{ij} \hat{X}_{ij} \quad (7)$$

$$\hat{X}_{ij} = \frac{X_j - X_i}{\|X_j - X_i\| + \epsilon} \quad (8)$$

$$\hat{K}_{ij} = \frac{K_i - K_j}{K^{worst} - K^{best}} \quad (9)$$

where K in general is a fitness value of a given krill, so K^{worst} and K^{best} are the worst and the best fitness degrees achieved so far by any individuals. NN is a number of reachable krill neighbours and ϵ is an immaterial positive number introduced to avoid singularities in the formula.

The NN value depends on a stated sensing scope of krill individuals. It can be defined in a static way, e.g. individuals always take into consideration a constant number of closest krills, regardless of their distance. The other way is to determine neighbour sets using the heuristic in every iteration:

$$d_{s,i} = \frac{1}{5N} \sum_{j=1}^N \|X_i - X_j\| \quad (10)$$

Every krill individual has its target vector defined as follows:

$$\alpha_i^{target} = C^{best} \hat{K}_{i,best} \hat{X}_{i,best} \quad (11)$$

where

$$C^{best} = 2 \left(rand + \frac{I}{I_{max}} \right) \quad (12)$$

I , I_{max} is the current iteration number and a maximum number of iterations. $rand$ is a random value between 0 and 1.

- F_i - food foraging:

$$F_i = V_f \beta_i + \omega_f F_i^{old} \quad (13)$$

where V_f is the food foraging speed and ω_f , as previously, is the inertia of the movement. Food fitness of the individual is defined as follows:

$$\beta_i = \beta_i^{food} + \beta_i^{best} \quad (14)$$

And food attraction for the krill individual is:

$$\beta_i^{food} = C^{food} \hat{K}_{i,food} \hat{X}_{i,food} \quad (15)$$

The food coefficient, expressing global attraction of the food center, is:

$$C^{food} = 2 \left(1 - \frac{I}{I_{max}} \right) \quad (16)$$

$$\beta_i^{best} = \hat{K}_{i,best} \hat{X}_{i,best} \quad (17)$$

where $K_{i,best}$ is the best fit achieved by given krill individual so far.

- D_i - random physical diffusion. In the simplest case it can be fully random. A general rule concerning the diffusion states that the better the krill's position is, the less its random motion diffusion is. The following formula defines a diffusion on the basis of that rule and the assumption that krills' positions improve by the time:

$$D_i = D^{max} \left(1 - \frac{I}{I_{max}} \right) \delta \quad (18)$$

where D^{max} is the maximum possible diffusion, δ is the random directional vector.

The distinctive feature of the KH algorithm is a implementation of two basic evolutionary operators, crossover and mutation, despite it is not really an evolutionary algorithm. The crossover operator is inspired by the genetic algorithms. Its result is an individual with some features of both "parents". Mutation, in turn, is a random change in an individual's features. Paper [7] contains a description of the process of incorporating mutation scheme into the KH algorithm. A stop condition for the algorithm could be a time limit, reaching a desired fitness level or the combination of these two. High level pseudocode of the KHA is presented as the Algorithm 2. Applications and studies on parameters tuning of the KHA have been described in publications: [8], [9], [10] and [11]. Papers [12] and [13] contain other proposed modification of the algorithm.

Algorithm 2 High-level KHA pseudocode

Define and populate algorithm's data structures

Initialize random initial population

while stop condition is reached **do**

 Evaluate fitness of each krill individual on the basis of its position

 Calculate motion of each krill individual

 Perform genetic operations

 Update each krill position in the search space according to calculated motion and eventual genetic operations results

end while

II. CODB - PROTOTYPE IMPLEMENTATION OF A COLUMN-ORIENTED DB

One of the main goals of the research is the implementation of a column-oriented database management system (the term CODB will be used). A typical contemporary Java SE development stack has been chosen as an implementation environment. It consists of the standard Java Development Kit 8 along with supplementary libraries Google Guava (general purpose library), Logback (logging facility), junit (unit testing framework) and Mockito (mocking facility for unit tests). This implementation is used as a foundation for the further research.

Implementation objectives were stated as follows:

- possibility to be used as embedded database in Java and other JVM-based programming languages, but enabling future use with custom connectivity protocol and drivers for other languages, as well as a REST service;
- usage of the memory-mapped data files, thus requiring 64-bit OS for sufficient performance;
- UUID v1 as an objects' identifiers;
- custom binary storage file structure (please see the description below);
- all values stored as an UTF-8 encoded strings;
- full in-memory indexing;
- static storage garbage collection;
- full unit test coverage of logic and storage code (except so called boilerplate code, such as field accessor methods);
- lack of any access rights/database user management facility;
- object-oriented API;
- partitioning and/or sharding support.

A. Storage structure

The CODB supports one keyspace in one running instance, i.e. one instance serves for only one keyspace. Keyspace is simply a directory in the operating system's file system tree. It does not have any metadata except a name. The keyspace directory contains subdirectories representing column families.

The column family directory contains binary column datafiles, one file per one column. Binary data files contain a sequence of value entries. Every value entry consists of the actual UTF-8 encoded value, a set of UUIDs of records which

contains a given value and length values necessary to calculate offsets. UUIDs are stored as a pair of long (64-bit) values. Every length value is just a long 64-bit value. Please note that every specific value is stored only once, notwithstanding the number of records which contain it. Table I shows a structure of a single entry in the column datafile.

The current implementation offers full in-memory indexing. A full datafiles scan is performed during startup in order to build the indices. Planned materialized indices or eventual column metadata would be stored in the same directory as the data. The indexing facility in the CODB engine has two aspects:

- indexing physical location (offset) of column values in the datafile;
- indexing offset of spare space fragments in the datafile (see description below).

Because of performance reasons record entries (that is pairs (value, UUID)) are not actually updated, but removed and created again with new values. When a value which belongs to a given record is changed, the record's identifier is removed from the UUID set assigned to the previous value and will be added into UUID set of a new value. If there is no entry of a new value, it is appended at the end of the datafile. If the identifier set of the new value does not have enough space to be extended directly in its place, it is removed from its current place and appended at the end of the datafile. When a value is removed from the last record which possessed it, the record's identifier is removed from the set but the value remains in the datafile. That feature raises a need of some garbage collecting mechanism. A prototype implementation offers a static garbage collection facility, that is a separate application which is intended to be run when the actual DBMS engine is not running. The garbage collector performs a full data scan to build the full index and then squashes the datafile by placing values one directly after another and throwing out the values without any corresponding records. That way, spacings introduced while updating values, as well as unused value entries, are removed from the datafiles.

III. MATHEMATICAL MODEL OF A COLUMN-ORIENTED DB AND APPLICATION

In order to apply selected natural computation algorithms to the CODB project, one needs to translate fundamental concepts of the domain into the mathematical model which reflects the nature domain. Correct translation between these three fundamentally different worlds is the key to obtaining satisfying results. Natural computation algorithms are a general heuristic mean of optimization problem solving. They are not tailored to any particular problem domain, but the problem itself should provide a mathematical model instead. Such models consist of a few basic components:

- objective function;
- input as a scalar or vector value or n-dimensional matrix;
- output as a scalar or vector value or n-dimensional matrix.

The objective function for the considered problem must take several criteria into consideration when calculating the score for the candidate partitioning solution, these include:

- partitioning scheme application cost, that is cost of moving data back and forth between database engine instances;
- estimated cost of the data division introduced by the selected partitioning scheme.

Input for the objective function is a candidate solution. Its output is the fitness degree, a value which determines a given candidate's quality. Enumerated objectives can be formulated on the basis of a query log and other data usage statistics, which must be analysed for mutual co-appearance of different data parts. The superior goal here is placing data which often occur together in the same instance.

The input of the algorithm should consider at least the following aspects:

- relationship between different data parts;
- current state of the database;
- cost of moving the data item between instances (usually size).

For every application, such a list can differ fundamentally, but probably always it will be an n-dimensional matrix, in which one of the dimensions will represent all records currently stored in the considered database.

The output of the algorithm is the most fit model of partitioning the database. In the simplest, non-optimized case, it will be at least a 2-dimensional matrix, in which one dimension covers all the records.

IV. SUMMARY

This paper summarises the overall perspective on the research about horizontal scalability of column-oriented databases with use of natural computing algorithms. The research has already brought some encouraging results. Currently, the biggest challenge is the creation of an appropriate objective function reflecting all the necessary aspects of distributed database operation. Correct mapping of the problem from the database domain to the mathematical model also is crucial for the effectiveness of the proposed solution. The primary goal at the current stage is careful creation of the model and elaboration of the database usage log analysis methods. Another important objective is inspection of the FPA and KHA algorithms' features and properties in details. Creation of the fully featured column oriented database management system in Java, intended as a foundation of solution application is yet another goal, although it goes beyond the scope of the research.

REFERENCES

- [1] A. Nowosielski, "RDBMS horizontal scalability – architectures review and example implementation," 2013, AGH University of Science and Technology, M.Sc.-Thesis.
- [2] D. Abadi, "The design and implementation of modern column-oriented database systems," *Foundations and Trends® in Databases*, vol. 5, no. 3, pp. 197–280, 2012. doi: 10.1561/1900000024. [Online]. Available: <http://dx.doi.org/10.1561/1900000024>

TABLE I
SINGLE ENTRY STORAGE STRUCTURE

| Data type | Size | Content |
|-----------------------------|----------------|---------------------------------|
| long | $8B$ | value length - l |
| UTF-8 encoded string | $\geq lB$ | actual value |
| long | $8B$ | number of associated keys - n |
| 0 or more (long,long) pairs | $n \times 16B$ | big endian encoded keys |

- [3] "Apache Cassandra™ 2.1 Documentation," 2014, (access 28th June 2015). [Online]. Available: <http://docs.datastax.com/en/cassandra/2.1/pdf/cassandra21.pdf>
- [4] A. H. Gandomi and A. H. Alavi, "Krill herd: A new bio-inspired optimization algorithm," *Communications in Nonlinear Science and Numerical Simulation*, vol. 17, no. 12, pp. 4831–4845, 2012. doi: 10.1016/j.cnsns.2012.05.010. [Online]. Available: <http://dx.doi.org/10.1016/j.cnsns.2012.05.010>
- [5] X.-S. Yang, "Flower pollination algorithm for global optimization," in *Unconventional Computation and Natural Computation*. Springer Science Business Media, 2012, pp. 240–249. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32894-7_27
- [6] X.-S. Yang, M. Karamanoglu, and X. He, "Multi-objective flower algorithm for optimization," *Procedia Computer Science*, vol. 18, pp. 861–868, 2013. doi: 10.1016/j.procs.2013.05.251. [Online]. Available: <http://dx.doi.org/10.1016/j.procs.2013.05.251>
- [7] G. Wang, L. Guo, H. Wang, H. Duan, L. Liu, and J. Li, "Erratum to: Incorporating mutation scheme into krill herd algorithm for global numerical optimization," *Neural Comput & Applic*, vol. 24, no. 5, pp. 1231–1231, 2013. doi: 10.1007/s00521-013-1422-y. [Online]. Available: <http://dx.doi.org/10.1007/s00521-013-1422-y>
- [8] S. Łukasik and P. A. Kowalski, "Study of flower pollination algorithm for continuous optimization," in *Intelligent Systems'2014*. Springer Science Business Media, 2015, pp. 451–459. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-11313-5_40
- [9] P. A. Kowalski and S. Łukasik, "Experimental study of selected parameters of the krill herd algorithm," in *Intelligent Systems'2014*. Springer Science Business Media, 2015, pp. 473–485. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-11313-5_42
- [10] G. P. Singh and A. Singh, "Comparative study of krill herd, firefly and cuckoo search algorithms for unimodal and multimodal optimization," *IJISA*, vol. 6, no. 3, pp. 35–49, 2014. doi: 10.5815/ijisa.2014.03.04. [Online]. Available: <http://dx.doi.org/10.5815/ijisa.2014.03.04>
- [11] P. K. Adhvaryyu, P. K. Chattopadhyay, and A. Bhattacharjya, "Application of bio-inspired krill herd algorithm to combined heat and power economic dispatch," in *2014 IEEE Innovative Smart Grid Technologies - Asia*. IEEE, 2014. doi: 10.1109/isgt-asia.2014.6873814. [Online]. Available: <http://dx.doi.org/10.1109/isgt-asia.2014.6873814>
- [12] L. Guo, G.-G. Wang, A. H. Gandomi, A. H. Alavi, and H. Duan, "A new improved krill herd algorithm for global numerical optimization," *Neurocomputing*, vol. 138, pp. 392–402, 2014. doi: 10.1016/j.neucom.2014.01.023. [Online]. Available: <http://dx.doi.org/10.1016/j.neucom.2014.01.023>
- [13] G.-G. Wang, A. H. Gandomi, and A. H. Alavi, "Stud krill herd algorithm," *Neurocomputing*, vol. 128, pp. 363–370, 2014. doi: 10.1016/j.neucom.2013.08.031. [Online]. Available: <http://dx.doi.org/10.1016/j.neucom.2013.08.031>