Chernishev *J Big Data* (2017) 4:5
DOI 10.1186/s40537-017-0069-4

Journal of Big Data

SURVEY PAPER Open Access

The design of an adaptive column-store system

George Chernishev*

*Correspondence:
chernishev@gmail.com
Saint-Petersburg University,
Saint Petersburg, Russia

Abstract

A fully self-managed DBMS which does not require administrator intervention is the ultimate goal of database developers. This system should automate deploying, configuration, administration, monitoring, and tuning tasks. Although there are some advances in this field, self-managed technology is largely not ready for industrial use and remains an active area of research. One of the most crucial tasks for such a system is automated physical design tuning. A self-managed approach for this task implies that the physical design of a database should be automatically adapted to changing workloads. The problems of materialized view and index selection, data allocation, horizontal and vertical partitioning were studied for a long time, and hundreds of approaches were developed. However, most of these approaches were static, thus, unsuitable for self-managed systems. In this paper we discuss the prospects of an adaptive distributed relational column-store. We show that the column-store approach holds a great promise for construction of an efficient self-managed database. At first, we present a short survey of existing physical design studies and provide a classification of approaches. In the survey, we highlight the self-managed aspects. Then, we provide some views on the organization of a self-managed distributed column-store system. We discuss its three core components: an alerter, a reorganization controller and a set of physical design options (actions) available to such a system. We present possible approaches for each of these components and evaluate them. Several physical design problems are formulated and discussed. This study is the first step towards a creation of an adaptive distributed column-store system.

Keywords: Column-stores, Physical design, Self-management, On-line tuning

Introduction

This paper is an extended version of the conference paper [1]. The era of big data poses new requirements for database management software. A contemporary DBMS should provide a basis for efficient storage and handling of big data. The aspects of big data support in DBMS can be classified into two groups:

- Performance-oriented aspects. Such a DBMS should be able to load, store and query big data efficiently.
- Management-oriented aspects. It should be easy for a database administrator to manage DBMS handling big data.

Recently, a novel type of DBMS, called column-store DBMS, has appeared. While its performance in regards to the first group of aspects is excellent, its ability to support the second group is an actual research problem.

Automatic database tuning is one of the oldest research problems in the database domain. It spans about forty years of evolution and can be described by the three following stages:

1. Separate attempts to automate a selection of various physical design structures: vertical partitioning [2–5], horizontal partitioning [6–9], data allocation [10–13], indexes [14–16] and materialized view selection [17]. These works usually consider only a single aspect of database physical design. They typically provide an algorithm or an idea not tied to a particular database system and do not employ the "what-if" mechanism. Instead, they propose their own cost models. This stage started in the 70s and largely ceased in the middle of the 90s when the next approach had emerged.
2. The development of advisors or recommenders. An advisor is a tool which recommends actions concerning database physical design using the knowledge of a data schema and a workload. This approach is characterized by the integration with some database system and the use of its query optimizer in the "what-if" mode. Moreover, the advisors often consider several physical design options simultaneously. Finally, unlike the previous class, these are not only algorithms, but full-fledged tools. They are aimed for industrial application and for the industrial end user. One of the first such tools is the REDWAR [18] tool for database analysis. Later, there were the AutoAdmin [19] tool for index recommendation, the DB2 advisor [20–23] (index, materialized view, partitions, allocation), Oracle Advisor [24, 25] (index, materialized view, partitioning support), various advisors for Microsoft's products [26–30] (variety of physical structures), PostgreSQL [31–33], Ingres [34]. This stage started at the end of the 80s and continues up to this day.
3. The self-management approach. In the majority of previous studies, it was the administrator who made the decision whether to apply the proposed database reorganization or not. Self-management approach aims for complete elimination of human intervention in the database tuning cycle. To the best of our knowledge, the first studies of this type appeared about ten years ago [35, 36]. It is essential to note that several systems (or their parts) mentioned above can be also considered as self-management ones.

A comprehensive survey of automatic physical design tuning, listing a large amount of works involving all the aforementioned types, can be found in reference [37].

Self-management technology aims to automate the tasks of deploying, configuration, administration, monitoring, and tuning of a database system to the largest extent possible [38]. The reason for the interest in self-managing systems is simple: it is widely accepted that the contribution of these tasks to the Total cost of ownership (TCO) of a database system is high. Some reports [29, 38–40] indicate that the TCO of a database system is dominated by human-related expenses, while others [41] claim that it is not true for very large scale systems.

In this paper, we discuss the self-management aspects of a distributed relational column-store database. A column-store database is a database which keeps each attribute in a separate file, unlike traditional row-stores, which use the slotted page layout [42]. This different storage model leads to different query processing schemes and subsequently to a different set of physical design options. Recently, several such systems [43, 44] emerged and quickly gained popularity in both academic and industrial communities due to their exceptional performance on read-only workloads. The unique properties of the column-stores hold a great promise for construction of an efficient self-managing system. In this paper, we evaluate the prospects of self-managed distributed column-store system development.

We propose the overall scheme of such a system which includes an alerter, a reorganization controller and a list of possible physical design options (actions). We start with the description of the environment and the alerter component. Next we present two possible design approaches for controller construction. Finally, we discuss the novel physical design options provided by the storage model, assess their impact on the physical design and meditate on the automating of their selection. We assume that the reader is familiar with the basics of column-store technology. Otherwise, we suggest the following surveys [44, 45].

To the best of our knowledge, this is the first work to consider on-line physical design tuning in a distributed column-store database.

Database (self-)tuning basics

The behavior of a database system can be described by the following formula [38]:

$$f : configuration \times workload \rightarrow performance$$

Here, the system configuration consists of a hardware setup (characteristics of the used hardware), software setup (boot-time parameters), database physical design and so on [38]. The workload characteristics include data schema information, query information (frequencies, attributes involved), and their arrival patterns. The performance is represented by one of the performance metrics (response time, throughput, reliability, etc.) or their combination.

Having a model involving all of these components, one may try to find the best configuration—the configuration maximizing a given performance metric [38]. In this paper we are interested solely in the physical design aspect of the configuration component. If we restrict configuration to a set of physical design structures, we will get the general formulation of the physical database tuning problem. Finding the best configuration even for a single type of structures is usually an NP-hard problem [5, 8, 46, 47]. Thus, a heuristic algorithm is required.

There are three popular methods for this kind of problems: integer programming [3, 33], a general heuristic algorithm [2, 4] and a domain-specific heuristic algorithm [20, 28, 30]. The latter is usually tightly coupled with a database system and relies on "what-if" calls.

Most of the works up to 2005 considered the physical design problem as a static problem, i.e. the workload cannot change after the selection of a configuration. However, a self-tuning system should be adaptive to the workload, thus a "dynamization" of the

algorithm is needed (usually the selection algorithm is too expensive to be called multiple times at will).

Here, the observe-predict-react cycle [38] comes into play. It is a general framework for construction of an adaptive database system. The observe component monitors specified workload characteristics, such as the time it takes to process a given query or the cardinality of a given relation. The predict component is used to assess performance of the current configuration in the near future and to compare it with other possible configurations. The react component is engaged when the current configuration is found unfit and a new configuration should be selected.

There are several dimensions of a self-managed database [48]: Self-Configuration, Self-Optimization, Self-Healing and Self-Protection. Automatic physical design tuning fits into the first three of them. These dimensions were partially implemented in systems of the past, but no fully self-managed system yet exists.

Now, let us review contemporary systems featuring self-management components.

Related work

On-line tuning in row-stores

In the recent years, there was a number of prototypes that employed the self-tuning approach for the physical configuration in row-stores. The Table 1 contains a short summary of the related works. In this table, the second column describes what type of physical design structure the study concerns and the third column shows what platform was used for the evaluation.

Continuous on-line tuning (COLT) [35] is a framework which adjusts the system configuration in order to maximize query performance with respect to the active query set. The proposed approach is to select the most beneficial indexes taking into account a storage budget. The authors implemented it using the PostgreSQL database system.

The reference [36] describes the alerter component of a self-tuning system. This component periodically checks whether there is a configuration which will result in a performance improvement. The alerter produces the lower and upper bounds of the improvement if a tuning component is run. Alerter component is designed for indexes, but the authors also describe its application for materialized views.

An on-line selection of aggregation tables is considered in the reference [49]. Similarly to the reference [36], the core component of this system is the alerter, which notifies the user and presents a beneficial configuration and a supposed cost reduction. It is interesting to note that the authors work not with a relational query language, but with the MDX query language. The prototype was implemented as an extension of Mondrian, an open source OLAP server.

AdaptPD [50] is an on-line tuning tool for vertical partitioning. The authors proposed a "cache-and-reuse" technique for query cost estimation. The idea is to cache the query plans that do not change across several configurations, thus reducing the number of optimizer calls. Another difference of this work is the use of asymmetric configuration transition costs. The authors use the SDSS astronomical database and Microsofts SQL Server for experimental validation.

An on-line index tuning approach taking user feedback into account was proposed in the reference [51].

Table 1 On-line tuning of row-stores, a summary

Study	Structures	Experiments	Notes
COLT [35]	Indexes	PostgreSQL	Uses storage budget constraint; employs "what-if" optimizer mode; manages active, hot and cold sets of indexes
Alerter [36]	Indexes, materialized views	SQL server	Uses storage budget and mininum improvement constraints; Notifies DBA and provides a set of candidate structures
Alerter [49]	Aggregation tables	Mondrian	Uses soft storage budget constraint
AdaptPD [50]	Vertical partitioning	SQL server	Employs "what-if" optimizer mode with caching
WFIT [51]	Indexes	IBM DB2 Express-C	Takes a workload and user feedback into account
MISO [52, 53]	Materialized views, storage selection	Multiple	Tuning of multistore system physical design. Uses storage and transfer constraints
EVO [54]	Indexes	Multiple	Authors propose query plan transformations using genetic algorithm for index selection
ARH [55]	Automatic re-indexing	PostgreSQL	A set of heuristics is used to decide when to trigger a re-indexing process to counter an index fragmentation
Tuner [33]	Multiple	PostgreSQL	An on-line tool which tunes several physical structures—indexes, partitions, and is capable of tracking index interaction
AutoStore [56]	Vertical partitioning	Custom	A comparison of on-line algorithms for vertical partitioning
SMOPD [57]	Vertical partitioning	Custom	Closed itemset mining for on-line vertical paritioning

The reference [52] describes on-line physical design tuning in a multistore system. A multistore system is a system which encompasses several different data stores (HDFS and RDBMS) and allows for simultaneous querying of data kept in all types of stores. Different types of data are stored in different systems, for example, HDFS can be used to keep big log files and RDBMS may contain analytical business data. The building block of the proposed physical design is the so-called opportunistic materialized view, which is the byproduct of query processing on a Hadoop-based system. The system automatically adapts to the dynamically changing workload. In order to achieve this, the authors tune the set of opportunistic materialized views in each store and solve the data placement problem. The same opportunistic materialized views are used for tuning of UDFs (user defined functions) in the reference [53].

An approach combining an evolutionary algorithm with a workload compression module is proposed in the reference [54]. In their work, the authors address the problem of on-line index selection. Each chromosome holds a vector of plans for each of the seed tasks (a query which passed through the compression module). The elements of a chromosome are dynamically added or removed, depending on a changing workload.

The self-healing aspect of a self-managing database was considered in the reference [55]. The authors developed a set of heuristics for the problem of index fragmentation. These heuristics control the re-indexing process.

Continuous on-line tuning was used in an on-line tuning tool for PostgreSQL described in the reference [33]. This tool is capable of recommending both indexes and partitions using a unified model and is capable of adapting to changes in the workload. Additionally, the tool recommends a beneficial order of index materialization. Several other interesting algorithms and techniques are incorporated in this tool.

On-line vertical and horizontal partitioning is considered in the reference [56]. Authors employ the idea of attribute affinity used in the works of 80s and 90s (e.g. [4, 5]) and "dynamize" it. The result is called AutoStore, an automatically and on-line partitioned database store.

SMOPD [57] uses closed item sets mining to perform an on-line vertical partitioning of a set of tables.

There are many more approaches which involve physical self-tuning in row-stores, but we are limited by the space to describe them all.

Database tuning for column-stores

Column-stores, on the other hand, being a much younger field of research had significantly less time to develop automatic tuners. Still, there are several relevant studies, a short summary is presented in the Table 2. Let us consider them.

Reference [58] describes on-line physical design tuning for the in-memory database SAP HANA [59]. This database is designed to handle both transactional and analytical workloads. The goal of the proposed physical design tuning is to select a more beneficial table storage mode: a column-store or a row-store. In order to provide such recommendations the authors developed a cost-based model. The next idea is store-aware partitioning which is as follows: split a table into different parts and keep them in different stores.

H_2O database system [60] proposes an on-line data reorganization with on-the-fly query compilation. The data reorganization is represented by a change of vertical partitioning schemes ranging from the row-store to the column-store.

Peloton [61] is an open-source in-memory adaptive DBMS designed for hybrid transaction-analytical processing. This system is able to adapt both vertical and horizontal partitioning schemes. It produces table layouts ranging from NSM to DSM and thus, it

Table 2 Tuning of column-stores, a summary

Study	Notes
SAP HANA [58]	In-memory, data reorganization
H_2O [60]	Query compilation and data reorganization
Peloton [61]	In-memory system, can adapt data layout
Vertica [62]	Projection recommender, not online
C-store [63]	Recommender of materialized views, not online
Cliffguard [64]	Robust configuration recommender, works with Vertica
Snowflake [65]	Commercial, distributed, relies not on tuning, but on data pruning

can be considered an adaptive column-store. For vertical partitioning, it uses a modified k-means algorithm. Unfortunately, Peloton is not a distributed system.

Vertica is a commercial distributed column-store database. It has an automatic physical design component [43] that helps to select a set of projections for a given storage budget. A comprehensive description of this system is presented in the reference [62]. However, this designer is not an on-line tool.

Another study related to physical design in column-stores is the reference [63]. In this paper, the problem of materialized view selection is considered. The proposed cost model takes into account sort orders and inserts. However, this physical design is intended for a centralized system, and it does not consider allocation. Also, this approach is a static one, not an on-line one.

Cliffguard [64] is an automatic physical design tuning tool which aims not for the best possible solution, but for a robust one. A robust solution is a solution that is robust against parameter uncertainties (parameter changes or bad estimates). Essentially, this approach allows to trade optimality for the desired level of robustness. The proposed tool uses robust optimization theory and is built to interact with any existing (non-robust) physical designer. Thus, it is able to recommend DBMS-specific structures. For example, authors used it to recommend projections in the Vertica column-store system.

Snowflake [65] is an another commercial system capable of column-oriented data processing for semi-structured data. It adopts a Software-as-a-Service model and aims to free user from complex management tasks. Thus, it discards the physical design component in favor of extensive data pruning.

We conclude our survey with the following:

- Currently, there is a heightened interest in on-line tuners in row-stores.
- There is a shortage of on-line tuning tools for column-store systems, especially for distributed disk-based ones.

Column stores

A column-store database is a system which keeps each attribute separately, as opposed to row-store systems. This leads to a number of conceptual differences. First of all, a classic approach to query engine construction—the Volcano model [66] needs to be modified. In column-stores, operators exchange not only data, but also positions (or IDs). Furthermore, column-stores require introduction of operators which process positions only. For example, consider Fig. 4a, the part with two DS1 and one AND1 operator. Each of the DS1 operators returns positions from the corresponding attribute which satisfy a given predicate. The AND1 operator calculates conjunction and returns the list of record IDs which satisfy both of these predicates. The second difference is the need for reconstruction joins. At some point of query plan positions have to be substituted with corresponding attribute values. This process is called materialization. It can be done in a straightforward (and expensive) way using joins or using some kind of a DBMS-specific technique.

Eventually, a number of architectural differences arise. Let us summarize them over several different column-store implementations:

- A new approach to query processing: the problem of early/late materialization (when to perform tuple reconstruction), novel query plans (in some approaches a query plan is not a DAG anymore) and cost models, a new algebra of operations.
- A new approach to operator design: new operators and their implementations are possible.
- Compression is ubiquitous in this class of systems.

See studies [44, 45, 67] for more detailed introduction.

There are several major research prototypes and a large number of commercial implementations [44, 45, 68]. Let us characterize the research ones briefly:

- C-store is a disk-oriented column-store database. It features different sort orders, compression (different compression method may be used for each column), late materialization, special join operators and operations on compressed data.
- MonetDB is a main memory-oriented column-store database. It puts a special emphasis on efficient hardware usage, e.g. minimization of CPU cache misses or utilization of hardware parallelism. It features a special algebra operating on columns, adaptive indexing technologies (index is a by-product of query execution) and operators designed for an efficient hardware usage.
- To address the shortcomings of MonetDB, a new main-memory system called MonetDB/X100 was developed. The main novelties is the presence of block/vector of a column processing and a cooperative scans.
- Supersonic [69] is an open-source in-memory columnar query engine which is oriented for efficient data processing. Its core features are cache consciousness, vectorized execution, instruction pipelining and SIMD usage.
- Peloton [61] is an open-source in-memory DBMS designed for hybrid transaction-analytical processing. This system has a built-in on-line tuning component.

All of these prototypes are local database systems, however there are commercial distributed implementations based on some of them. Also, there are academic attempts to "distribute" some of these prototypes, for example, via a middleware approach.

The design of a self-managed distributed column-store system

Problem, environment and queries

We propose to construct a distributed adaptive column-store system which uses a single column as a minimum unit of data storage. We consider starting from the classic formulation of the physical design tuning problem [38]: given a workload, data scheme, available hardware and, possibly, a set of user constraints (e.g. storage bound of each node) find a configuration consisting of physical design structures which maximizes the throughput of the system.

Environment

We consider the following environment. There is a set of nodes in a distributed column-store system. Each of them has its own hardware characteristics: available disk space, processing power, network link capacity and so on. Each node stores a number of

columns or their parts and is capable of performing not only scans, but complex operators like joins and aggregations.

Queries

The queries and their characteristics (frequencies, involved attributes, selectivities of their predicates) are used to control the reorganization.

The general scheme of a self-managed database system is presented in Fig 1. It has three core components: alerter, reorganization controller and a set of actions. Let us discuss them.

Alerter

The first problem is what we should adapt to, and what kind of information should use the alerter component. There are several possible approaches:

- The most common approach is to use a sliding window, which helps to keep track of the recently processed queries. These recent queries allow us to compute some aggregate characteristics, which are then used to decide whether to trigger reorganization or not.
- Another approach is to use the knowledge of some predefined query patterns. For example, we might know the existence of query patterns like it is done in the reference [70]. We can detect these patterns or rely on some external knowledge (e.g. we know that data loading happens only at night).
- A single query information can also be useful. Consider a query which takes a long time to complete. We can predict its performance and evaluate its plan. Then, we can start the evaluation and simultaneously start the physical reorganization. Later, we switch its evaluation to a new plan while reusing the partial results obtained earlier. The goal is to change it in such a way that the query would benefit from the reorganization on upcoming stages of processing.

Control of the reorganization

One of the key points of a self-managed database system is the control of the reorganization. We can propose the following approaches:

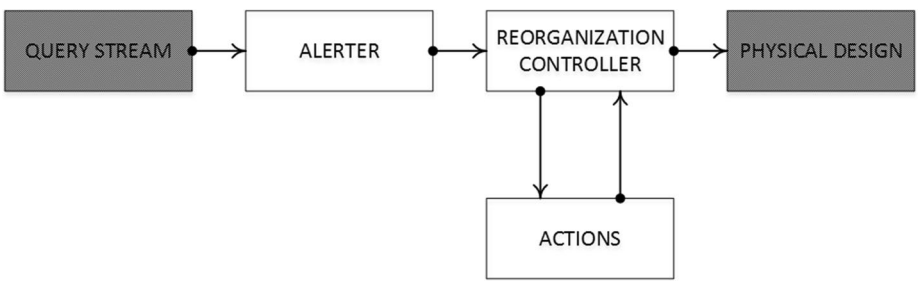

Fig. 1 The core components of a DBMS self-management system

- A cost-based optimization for a particular query or a set of queries. This is the classic approach used since 70s in the area of physical design tuning. Nowadays, it is the mainstream approach used almost in every industrial database [21, 28, 30, 33, 70] and in a majority of academic studies.
- A kind of heuristic strategy which will guide the search behavior of a system. This "forgotten" approach was also used since earlier days of physical design tuning [37]. In this approach, no performance model is used; the process is guided by a rule set. It was employed as a separate algorithm or as a pre-filtering step in later cost-based studies [28] in order to lower computational complexity of the problem. Another prominent example is the group of affinity-based approaches [4, 37].
- A combination of both.

In spite of the evident fact that the cost-based approach is superior in terms of the quality of produced recommendations, a strategy may have several strong points over cost-based optimization:

1. Firstly, a cost-based enumeration may be very expensive in terms of computation resources. Thus, a continuous re-run of the optimization routines is impossible. One may have to resort to query plan caching schemes or other types of result reuse. However, the application of this approach is hindered by the following considerations.
2. Not all queries may be known in advance. A good self-tuning system should be capable to cope with such a situation. Using a cost-based approach, we may not be able to decide on any required action at all. At the same time, a strategy may offer a reasonable action before the arrival of such a query, employing some rational assumptions regarding the data distribution.
3. Not all queries may be run. We may optimize a workload which is not run, thus wasting precious time and possibly harming a future workload.
4. Special use cases, such as incremental data loading. In this case, some sort of a strategy like "create a new instance by mirroring the existing columnar layout of a particular node" can be employed.
5. Estimation errors (or even absence of the statistics at hand), which will lead to a drop in the quality of recommendations.

Eventually, we should adopt an approach combining elements of cost-based and strategy approaches. Let us now consider what a column-store system can do to adapt to changing workloads.

Actions

There is a number of actions which alter a column-store system's physical design and provide benefits for different queries. One can regard these action and their results as the building blocks of a physical design for column store database. For example: replicate, relocate and set up an adaptive index structure on some column. Let us discuss these actions and compare them to their classic counterparts if they are available.

Column relocation

A relocation of a column or a set of columns from one processing node to another. This action may be taken basing on the "what-if" estimation for a particular query or according to some strategy. In this case a strategy might look like "eliminate or minimize inter-node communication for a given query". This option existed since the earliest distributed databases, relations were shipped around the network and allocated on the nodes. However, in the column-store case, there is a number of benefits for query processing and physical design:

- Faster relocation;
- Incremental relocation;
- Cheap vertical partitioning;
- Corrective query processing;
- Additional candidate configurations for physical design.

Fast relocation

In columns-stores relocation takes less time due to compression, a technique which column-store databases are particularly good at. Column-stores feature compression rates up to 1:10, while row-stores usually provide a 1:3 rate [67].

Incremental relocation

The whole relation can be incrementally relocated on a per-column basis. There are two approaches to relocation:

- Perform column deletion when all columns of the relation were copied.
- Perform column deletion right after it was copied. In this case, we immediately obtain a chunk of free space which can be used in a number of ways. For example, the node can start receiving columns of another relation.

Both of these strategies allow to run queries on partially relocated data. The first one is straightforward: both the receiving and source nodes can be used for query processing during the relation relocation. The receiving node can execute queries which involve already moved data. The second strategy also allows to run queries, but in a different way. Consider the following example presented in Fig. 2. There is a relation presented in a column-store form which consists of five compressed attributes. It is being relocated from one node to another using the immediate deletion strategy.

There are three stages in this process:

St 1: Start of relocation, attribute "A" is being moved to Node 2;

St 2: Attribute "A" was relocated to Node 2, attribute "C" is being moved. At this stage, the system can run all kinds of queries employing both nodes (**distributed query processing**);

St 3: At this stage, the system can run some queries **locally**. For example, query requesting attributes "A" and "C" can be processed on Node 2 and query requesting attributes "B" and "E" can be processed on Node 1. The system can also run all kinds of queries employing both nodes.

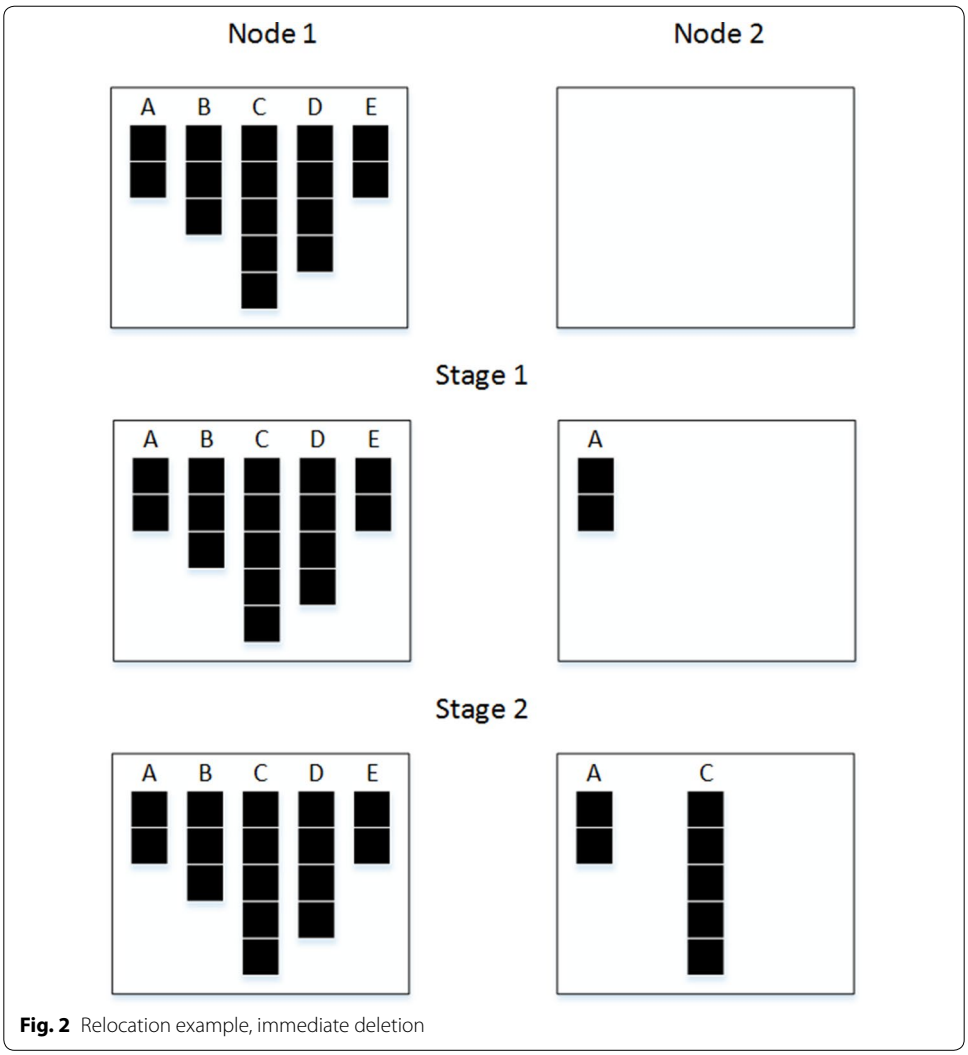

Fig. 2 Relocation example, immediate deletion

Some of the row-store systems can also process queries in the middle of relocation [71, section 5.3] (redefinition of partitions). However, it requires sophisticated algorithms to run queries in the middle of the relation relocation process. Also, the applicability and efficiency of these techniques is limited. Data layout of column-stores holds promise for overcoming these deficiencies.

We can manipulate the relocation process in order to minimally affect the performance of running queries. The column-stores offer new opportunities and promise to enhance the already-known ones. The relocation can be halted for a short time when a high-priority query comes in. It also can be postponed for a long time to avoid relocation during peak hours. A novel possibility is adjusting relocation, e.g. postpone the relocation of not all, but some of the columns in order to obtain query processing benefits in between. Another idea is to make relocation piggyback on the query reading the same attribute, in a similar manner to the study [72]. Thus, one can reduce the negative effects of interference between query execution and the relocation process to some extent.

In column-stores one can manage partial results of relocation—the columns which had been already moved. It is possible to manipulate column relocation order to increase

the "usefulness" of the relocated column set. For example, one can move "hot" columns (the ones which are frequently needed) first, if the halt of relocation can happen. On-the-fly adjustment of relocation orders is also possible.

Moreover, for query batch processing, one can devise a relocation schedule for a given relation which will further minimize the interference and query processing costs. Consider the next example presented in Fig. 3. Suppose that we have to execute the following read-only queries: AC, BD, ACD, BE, CDE. The reorganization process is happening and the starting configuration is the same as used on the last step of the previous example. Here, for the sake of simplicity, we assume that the network communication cost dominates all other costs.

This schedule is superior to other alternatives:

- Execute without taking relocation into account. This alternative would require too much of extra network communication.
- Halt relocation, execute queries, resume relocation. This also requires too much network communication.
- Finish relocation, then execute. In this case no inter-node parallelism is possible and query response time is large.

All these new techniques can be developed based on the cost models.

Cheap on-line vertical partitioning

Vertical partitioning is a tool which allows to drastically improve performance of a relational database. This physical design option is a well-studied research topic, but the proposed solutions are mostly static. Dynamic approaches started to appear only recently [56, 57], due to the following reason. In a classic database it would involve a whole scan of the original relation (all attributes would be scanned) and a formation of two new ones. Thus, frequent repartitioning would be rather expensive in a classic case.

The opportunity to move columns for a lesser cost will allow us to perform on-line vertical partitioning for column-stores. Both distributed and local (via column reorderings, see next subsection) vertical partitioning schemes are possible. Thus, it will allow to achieve a rapid response to workload changes and, in its turn, to improve the adaptivity of the system.

New aspects of corrective query processing (CQP)

Corrective query processing [73, section 7.1] is a technique used to correct query optimizer errors on the fly by replacing the current plan without discarding partial results. This study presents an example where query execution consists of three phases: initial, re-optimized and stitch-up. During the initial phase a query monitor detects that assumptions which led to selection of initial plan do not hold. Then, the re-optimized plan is calculated and run on the remaining data. Finally, both results are glued together.

The same idea can be employed in column-stores. For example, consider the following query:

```
 SELECT shipdate, linenum FROM lineitem WHERE shipdate<
CONST1 AND linenum < CONST2
```

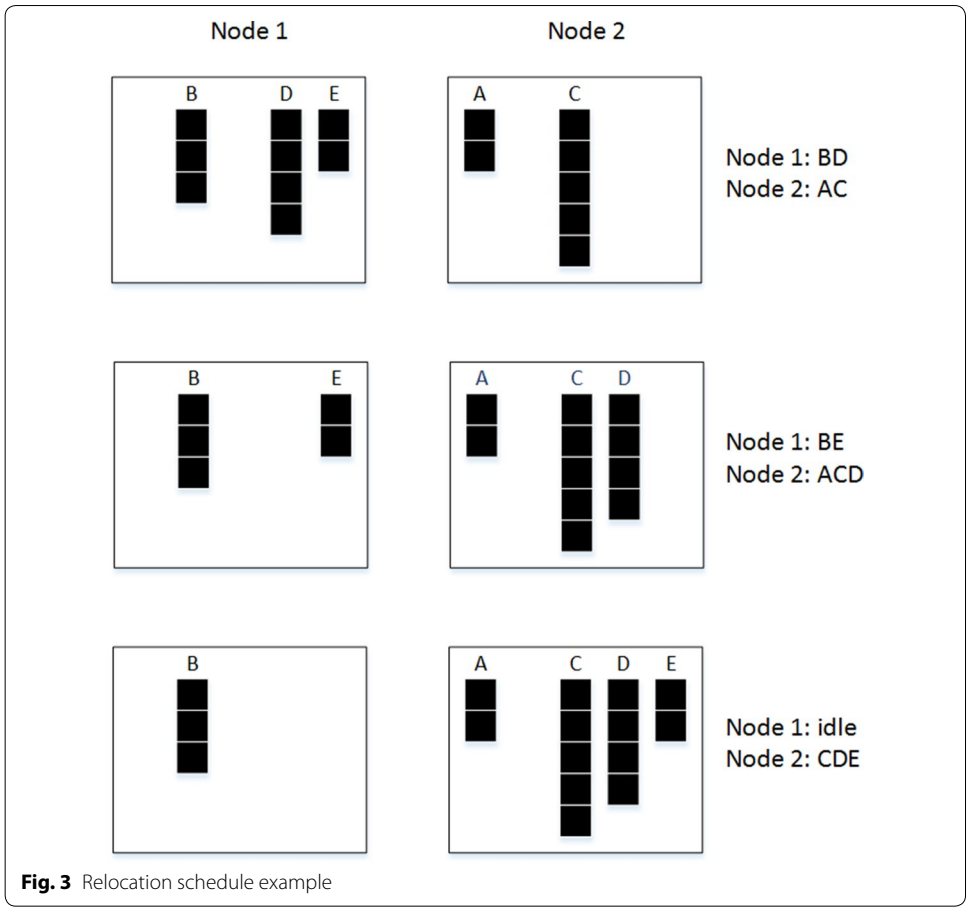

Fig. 3 Relocation schedule example

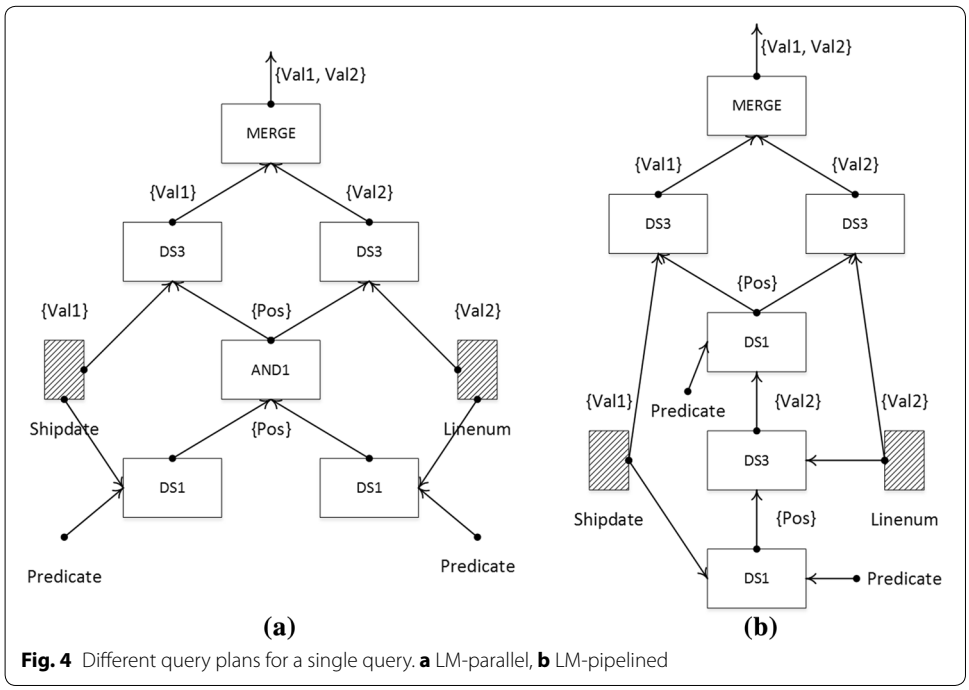

Fig. 4 Different query plans for a single query. **a** LM-parallel, **b** LM-pipelined

There are two possible execution plans. They are shown in Fig. 4 (the figure and query are adapted from the reference [74]). It is possible to perform on-line switching from plan depicted on the top to the plan on the bottom, if filtering Shipdate's column produces too many results. In this case, "AND1" operator should be substituted by a sequence of "DS3" and "DS1" operators. If filtering produces a low number of results, on-line switching in the opposite direction is also possible.

Existing approaches involve classic databases and the horizontal partitioning approach. Horizontal partitioning comes from the division of original data into initial and "reoptimized" parts. In column-stores one may try to come up with a different class of adaptation schemes, where reorganization happens after the processing of the whole column. This approach will allow to eliminate the stitch-up phase. In the same example it is possible to finish filtering the whole Shipdate's column before moving to another plan.

The query processing specifics, namely a large number of operators in a plan and small column sizes give hope that this kind of processing will perform well or will be profitable in terms of resources.

The aforementioned approaches are essentially query processing adaptivity. The entity which adapts is the query plan. There is also a place for physical design adaptivity too: adapt not only the query plan, but also the required physical structures. Sufficiently large query graphs and well-compressed columns can make this kind of an approach viable.

One of the promising scenarios of CQP in distributed environment is the following. Having a query, one must compute two query plans. The first plan is the best plan which can be run immediately, using existing physical design structures. The second query plan involves physical design structures which would be constructed during the query processing. The query execution is performed as follows: start the query processing and physical redesign simultaneously. Having finished redesign, move processing to the second plan. In order to make this scheme profitable, one must create a cost model which involves costs of physical redesign, plan switching and running both plans.

To the best of our knowledge, both of these kinds of column-store adaptivity were not considered in literature.

Additional candidate configurations for physical design

Column-stores have a much larger configuration space compared to row-stores. This contributes to overall flexibility of physical design, which leads to an increase in the number of beneficial configurations and to an improvement of the performance of the whole system.

All these properties will greatly contribute to the adaptivity of the considered DBMS.

Column reorderings

A reordering of a set of columns is a specialization for a query or a group of queries. The idea is to reorder the contents of a set of columns needed for a particular query according to the ordering of some column set (usually the most selective predicate's column). This option was described in the original C-Store paper [75] and in the Star Schema Benchmark [76, section 3.1] (multiple sort orders). Consider the following examples (the query notation is adopted from the reference [75]):

Table 3 contains the original data, which is represented by a relation with four attributes. This table is clustered by its primary key, id. This data can be considered as the following sort order: (id, name, wage, skill|id).

Next, Table 4 describes the sort order for the full wage query. The goal of this query is to retrieve records using the wage value as the key. The name "full" means that all original attributes would be requested. Note that all other attributes (columns) are sorted according to this attribute. This sort order is denoted as follows: (id, name, wage, skill|wage).

Finally, Table 5 contains the sort order for query (name, wage, skill|skill, name). This query uses skill and name as the key and omits the id column.

In these papers, sort orders were described as a query speed-up technique, while we can propose several sort order selection problems:

- Sort Order Selection Problem (SOSP): find a single good sort order for a given workload;
- Multiple Sort Order Selection Problem (MSOSP): find a good set of sort orders for a given workload and a given storage budget;
- Dynamization of SOSP and MSOSP: find a good set of sort orders for a dynamically changing workload and a given storage budget. This formulation considers the physical design problem in a dynamic environment, where query patterns change over time. The system should adapt stored sort orders to provide the best perfor-

Table 3 Source data

Id	Name	Wage	Skill
1	Ivan	80	C++
2	Petr	50	C++
3	Slava	30	Java
4	Vasya	60	Php
5	Sasha	70	Java

Table 4 Sort order for full wage query

Id	Name	Wage	Skill
3	Slava	30	Java
2	Petr	50	C++
4	Vasya	60	Php
5	Sasha	70	Java
1	Ivan	80	C++

Table 5 Sort order for (name, wage, skill|skill, name) query

Name	Wage	Skill
Ivan	80	C++
Petr	50	C++
Sasha	70	Java
Slava	30	Java
Vasya	60	Php

mance. While two previous formulations were already mentioned or even addressed in the past [63, 75], to the best of our knowledge, there are no studies which consider dynamic formulation for column-stores.

This option has no direct counterparts in the classic approach. It can be emulated by replicating the whole table and, thus, is very costly and its application is limited.

Column duplication

A creation (deletion) of a single column copy or a copy of a column set. The idea is to create a copy of some column transparently to the user, possibly modify it by other actions and then employ it to speed up query processing. If the column is not needed anymore, e.g. the cost-based estimator shows a more beneficial configuration or a new strategy prescribes to drop it — it could be deleted.

This copy can be used during construction of a sort order or in a standalone manner. Column-store technology offers the same benefits for this action: fast and incremental duplication of a relation, scheduling, CQP adaptation, and an increase of the number of useful configurations.

There is no direct analogue in the classic database systems. Most studies consider replication of the whole relation or its horizontal part. The closest approach is the creation of a materialized view; however, to the best of our knowledge, there were no studies for single-columnar views. Though the use of materialized views was considered for emulation of the column-store database in a comparison [77].

Horizontal partitioning

Horizontal partitioning of a column and partition merge. Horizontal partitioning of a relational database is a very well-studied problem, and there are many approaches to it [37]. One may consider value-based partitioning and non-value-based [78] (range or round-robin, for example), and primary and derived [6]. The column-store horizontal partitioning has some advantages over its counterpart in classic systems:

- The partition can be constructed for a lesser cost in terms of the number of accessed pages. Thus, one can achieve higher repartitioning speed compared to row-stores. This speed will provide a substantial benefit to an on-line tuning system, allowing it to adapt faster. Compression can be a potential source of problems, but the light-weight compression methods [43, 44] employed by the column-stores should not present any difficulties.
- The control of partitioning granularity—one can perform a partitioning of a subset of columns. For example, we can horizontally partition the "fat" column only without touching the rest. However, there could be queries which would benefit from partitioning columns together.
- The enlargement of the configuration space and the increased number of useful configurations lead to improved performance of query processing.

If the partitioning is not needed anymore, one can collapse a set of partitions. This action and the creation of a copy constitute a materialized view for a query, and vice versa. The

action has no direct counterpart in classic databases, but it can be emulated. However, it will require vertical partitioning or materialized view creation and thus, it is very costly.

Column-store specific actions

- Adaptive indexing. Adaptive indexing is a technique which constructs indexes on-the-fly during query processing. This approach is extensively used in column-store systems [44, 79–81]. To the best of our knowledge, there is only one study [82] related to selection and management of such indexes. Moreover, employing these indexes in a distributed environment looks promising because there are a lot of potential scenarios which may benefit from this structure. For example, one can keep two copies of a column on different nodes and perform a reorganization without any overhead for the processing. This can be achieved by alternating working and idle nodes. The application of these indexes in a distributed context also was not studied.
- Join index [75, section 2]. This is a structure used to speed up the reconstruction of a tuple by linking parts of a single record in a different projections. The follow-up paper [43, section 3.2] stated that early experiments showed that this technique was inefficient due to several reasons. However, in case of an adaptive system these results should be reconsidered since there are several novel scenarios. Unfortunately, there are no published studies regarding these experiments.

As we can see, a lot of actions share the same idea with classic relational databases. However, due to the different query processing (see [44]) scheme and the specifics of the data layout novel cost-based models are required.

Conclusion

In this paper, we discussed the design of a self-managed distributed column-store system. Architectural novelties of a column-store system hold a great promise for construction of an efficient self-managed database that would adapt its physical design to changing workloads. We surveyed state-of-the-art self-managed database systems and provided an overview of contemporary results. Next, we presented some thoughts on the organization of a self-managed distributed column-store system. We discussed the three core components: an alerter, a reorganization controller and a set of physical design options (actions) available to such a system. We described three approaches for the design of the alerter and two for the reorganization controller. Several physical design problems were formulated and discussed. Finally, we discussed available physical design options—the building blocks of an adaptive distributed column-store system.

Authors' contributions
All authors read and approved the final manuscript.

Acknowledgements
None.
This paper is an expanded version of the study [1].

Competing Interests
The authors declare that they have no competing interests.

Received: 10 December 2016 Accepted: 15 March 2017
Published online: 23 March 2017

References

1. Chernishev, G.: New trends in databases and information systems: ADBIS 2015 short papers and workshops, BigDap, DCSA, GID, MEBIS, OAIS, SW4CH, WISARD, Poitiers, France, September 8–11, 2015. In: Proceedings, chap. Towards self-management in a distributed column-store system. Cham: Springer; 2015. p. 97–107. doi:10.1007/978-3-319-23201-0_12.
2. Hammer M, Niamir B. A heuristic approach to attribute partitioning. In: Proceedings of the 1979 ACM SIGMOD international conference on Management of data, SIGMOD '79. New York: ACM; 1979. p. 93–101. doi:10.1145/582095.582110.
3. Hoffer JA. An integer programming formulation of computer database design problems. Inf Sci. 1976;11:29–48.
4. Navathe S, Ceri S, Wiederhold G, Dou J. Vertical partitioning algorithms for database design. ACM Trans Database Syst. 1984;9:680–710. doi:10.1145/1994.2209.
5. Lin X, Orlowska M, Zhang Y. A graph based cluster approach for vertical partitioning in database design. Data Knowl Eng. 1993;11(2):151–69. doi:10.1016/0169-023X(93)90003-8.
6. Ceri S, Negri M, Pelagatti G. Horizontal data partitioning in database design. In: Proceedings of the 1982 ACM SIGMOD international conference on Management of data, SIGMOD '82. New York: ACM; 1982. p. 128–36. doi:10.1145/582353.582376.
7. Ceri S, Navathe S, Wiederhold G. Distribution design of logical database schemas. IEEE Trans Softw Eng. 1983;4:487–504. doi:10.1109/TSE.1983.234957.
8. Sacca D, Wiederhold G. Database partitioning in a cluster of processors. ACM Trans Database Syst. 1985;10(1):29–56. doi:10.1145/3148.3161.
9. Bellatreche L, Woameno KY. Dimension table driven approach to referential partition relational data warehouses. In: Proceedings of the ACM twelfth international workshop on data warehousing and OLAP. New York: ACM; 2009. p. 9–16. doi:10.1145/1651291.1651294.
10. Copeland G, Alexander W, Boughter E, Keller T. Data placement in Bubba. In: Proceedings of the 1988 ACM SIGMOD international conference on Management of data. New York: ACM; 1988. p. 99–108. doi:10.1145/50202.50213.
11. Ghandeharizadeh S, DeWitt DJ. Hybrid-range partitioning strategy: a new declustering strategy for multiprocessor database machines. In: Proceedings of the 16th International Conference on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers Inc.;1990. p. 481–92.
12. Ghandeharizadeh S, DeWitt DJ, Qureshi W. A performance analysis of alternative multi-attribute declustering strategies. SIGMOD Rec. 1992; 21(2): 29–38. doi:10.1145/141484.130293.
13. Wong E, Katz RH. Distributing a database for parallelism. In: SIGMOD Record. 1983; 13(4): 23–29. doi:10.1145/971695.582201.
14. Hammer M, Chan A. Index selection in a self-adaptive data base management system. In: Proceedings of the 1976 ACM SIGMOD international conference on management of data. New York: ACM; 1976. p. 1–8. .
15. Stonebraker M. The choice of partial inversions and combined indices. Int J Paral Progr. 1974;3(2):167–88. doi:10.1007/BF00976642.
16. Bellatreche L, Boukhalfa K. Yet another algorithms for selecting bitmap join indexes. In: Bach Pedersen T, Mohania M, Tjoa A, editors. Data warehousing and knowledge discovery, *Lecture Notes in Computer Science*.Berlin : Springer; 2010. doi: 10.1007/978-3-642-15105-7_9.
17. Mami I, Bellahsene Z. A survey of view selection methods. SIGMOD Rec. 2012;41(1):20–9. doi:10.1145/2206869.2206874.
18. Yu PS, Chen MS, Heiss HU, Lee S. On workload characterization of relational database environments. IEEE Trans Softw Eng. 1992;18(4):347–55. doi:10.1109/32.129222.
19. Chaudhuri S, Narasayya V. Self-tuning database systems: a decade of progress. In: Proceedings of the 33rd international conference on very large data bases. VLDB Endowment. 2007. p. 3–14.
20. Rao J, Zhang C, Megiddo N, Lohman G. Automating physical database design in a parallel database. In: Proceedings of the 2002 ACM SIGMOD international conference on management of data, SIGMOD '02. New York: ACM; 2002. p. 558–569. doi:10.1145/564691.564757.
21. Zilio DC, Rao J, Lightstone S, Lohman G, Storm A, Garcia-Arellano C, Fadden S. DB2 design advisor: integrated automatic physical database design. In: Proceedings of the thirtieth international conference on very large data bases - vol. 30, VLDB '04, . VLDB Endowment. 2004. p. 1087–97.
22. Valentin G, Zuliani M, Zilio D, Lohman G, Skelley A. DB2 advisor: an optimizer smart enough to recommend its own indexes. In: Proceedings of 16th International Conference on Data engineering. 2000. p. 101–10. doi:10.1109/ICDE.2000.839397.
23. Zilio D, Zuzarte C, Lightstone S, Ma W, Lohman G, Cochrane R, Pirahesh H, Colby L, Gryz J, Alton E, Valentin G. Recommending materialized views and indexes with the IBM DB2 design advisor. In: Proceedings international conference on autonomic computing. 2004. p. 180–7. doi:10.1109/ICAC.2004.1301362.
24. Eadon G, Chong EI, Shankar S, Raghavan A, Srinivasan J, Das S. Supporting table partitioning by reference in oracle. In: Proceedings of the 2008 ACM SIGMOD international conference on management of data, SIGMOD '08. New York: ACM; 2008. p. 1111–22. doi:10.1145/1376616.1376727.
25. Dageville B, Das D, Dias K, Yagoub K, Zait M, Ziauddin M. Automatic SQL tuning in oracle 10g. In: Proceedings of the thirtieth international conference on very large data bases - Volume 30, VLDB '04. VLDB Endowment 2004. p 1098–9.
26. Bruno N, Chaudhuri S. Automatic physical database tuning: a relaxation-based approach. In: Proceedings of the 2005 ACM SIGMOD international conference on management of data, SIGMOD '05. p. 227–38. New York: ACM; 2005. doi:10.1145/1066157.1066184.
27. Agrawal S, Chaudhuri S, Kollar L, Marathe A, Narasayya V, Syamala M. Database tuning advisor for Microsoft SQL server 2005: demo. In: Proceedings of the 2005 ACM SIGMOD international conference on management of data, SIGMOD '05. New York: ACM; 2005. p. 930–32. doi:10.1145/1066157.1066292.
28. Agrawal S, Narasayya V, Yang B. Integrating vertical and horizontal partitioning into automated physical database design. In: Proceedings of the 2004 ACM SIGMOD international conference on Management of data, SIGMOD '04. ACM, New York, 2004. p. 359–70. doi:10.1145/1007568.1007609.

29. Agrawal S, Chaudhuri S, Kollar L, Marathe A, Narasayya V, Syamala M. Database tuning advisor for Microsoft SQL Server 2005. In: Proceedings of VLDB, 2004. p. 1110–21.
30. Nehme R, Bruno N. Automated partitioning design in parallel database systems. In: Proceedings of the 2011 international conference on management of data, SIGMOD '11. New York: ACM; 2011. p. 1137–48. doi:10.1145/1989323.1989444.
31. Gebaly KE, Aboulnaga A. Robustness in automatic physical database design. In: Proceedings of the 11th international conference on extending database technology: advances in database technology, EDBT '08. New York: ACM; 2008. doi:10.1145/1353343.1353365.
32. Maier C, Dash D, Alagiannis I, Ailamaki A, Heinis T. PARINDA: an interactive physical designer for PostgreSQL. In: Proceedings of the 13th international conference on extending database technology, EDBT '10. New York: ACM; 2010. p. 701–4. doi:10.1145/1739041.1739131.
33. Alagiannis I, Dash D, Schnaitter K, Ailamaki A, Polyzotis N. An automated, yet interactive and portable DB designer. In: Proceedings of the 2010 ACM SIGMOD international conference on management of data, SIGMOD '10. New York: ACM; 2010. p. 1183–6. doi:10.1145/1807167.1807314.
34. Thiem A, Sattler KU. An integrated approach to performance monitoring for autonomous tuning. In: Proceedings of the 2009 IEEE international conference on data engineering, ICDE '09. Washington: IEEE Computer Society; 2009. p. 1671–78. doi:10.1109/ICDE.2009.142.
35. Schnaitter K, Abiteboul S, Milo T, Polyzotis N. Colt: continuous on-line tuning. In: Proceedings of the 2006 ACM SIGMOD international conference on management of data, SIGMOD '06. New York: ACM; 2006. p. 1183–6. p. 793–5. doi:10.1145/1142473.1142592..
36. Bruno N, Chaudhuri S. To tune or not to tune?: A lightweight physical design alerter. In: Proceedings of the 32nd international conference on very large data bases, VLDB '06. VLDB Endowment .2006. p. 499–10.
37. Chernishev G. A survey of dbms physical design approaches. In: SPIIRAS proceedings. 2013; 24:222–76. http://www.mathnet.ru/trspy580.
38. Chaudhuri S, Weikum G. Self-management technology in databases. In: L. Liu M. Özsu, editors. Encyclopedia of database systems. Berlin: Springer; 2009. pp. 2550–5. doi:10.1007/978-0-387-39940-9_334DOIurl.
39. Kwan E, Lightstone S, Storm A, Wu L. Automatic configuration for ibm db2 universal database: compressing years of performance tuning experience into seconds of execution. In: Tech rep: performance technical report. New York: International Business Machines Corporation; 2002.
40. Dageville B, Dias K. Oracle's self-tuning architecture and solutions. IEEE Data Eng Bull. 2006;29(3):24–31.
41. Aboulnaga A, Salem K. Report: 4th int'l workshop on self-managing database systems (smdb 2009). 2009. p. 2–5.
42. Ailamaki A, DeWitt DJ, Hill MD, Skounakis M. Weaving relations for cache performance. In: Proceedings of the 27th international conference on very large data bases, VLDB '01. San Francisco: Morgan Kaufmann Publishers Inc.;2001 . p. 169–80.
43. Lamb A, Fuller M, Varadarajan R, Tran N, Vandiver B, Doshi L, Bear C. The vertica analytic database: C-store 7 years later. Proc VLDB Endow. 2012;5(12):1790–801. doi:10.14778/2367502.2367518.
44. Abadi D, Boncz P, Harizopoulos S. The design and implementation of modern column-oriented database systems. Hanover: Now Publishers Inc.; 2013.
45. Chernishev G. Physical design approaches for column-stores. SPIIRAS Proc. 2013;30:204–22.
46. Piatetsky-Shapiro G. The optimal selection of secondary indices is np-complete. SIGMOD Rec. 1983;13(2):72–5. doi:10.1145/984523.984530.
47. Harinarayan V, Rajaraman A, Ullman JD. Implementing data cubes efficiently. SIGMOD Rec. 1996;25(2):205–16. doi:10.1145/235968.233333.
48. Holze M, Ritter N. Towards workload shift detection and prediction for autonomic databases. In: Proceedings of the ACM First Ph.D. Workshop in CIKM, PIKM '07. New York: ACM; 2007. p. 109–116. doi:10.1145/1316874.1316892.
49. Hose K, Klan D, Marx M, Sattler KU. When is it time to rethink the aggregate configuration of your olap server? Proc VLDB Endow. 2008;1(2):1492–5. doi:10.14778/1454159.1454210.
50. Malik T, Wang X, Dash D, Chaudhary A, Ailamaki A, Burns R. Adaptive physical design for curated archives. In: Proceedings of the 21st International Conference on Scientific and Statistical Database Management, SSDBM 2009. Berlin: Springer; 2009. p. 148–66. doi:10.1007/978-3-642-02279-1_11.
51. Schnaitter K, Polyzotis N. Semi-automatic index tuning: keeping dbas in the loop. Proc VLDB Endow. 2012;5(5):478–89. doi:10.14778/2140436.2140444.
52. LeFevre J, Sankaranarayanan J, Hacigumus H, Tatemura J, Polyzotis N, Carey MJ. Miso: souping up big data query processing with a multistore system. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data, SIGMOD '14. New York: ACM, 2014. p. 1591–602. doi:10.1145/2588555.2588568.
53. LeFevre J, Sankaranarayanan J, Hacigumus H, Tatemura J, Polyzotis N, Carey MJ. Opportunistic physical design for big data analytics. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data, SIGMOD '14. New York: ACM, 2014. p. 851–62. doi:10.1145/2588555.2610512.
54. Kolaczkowski P, Rybinski H. Online index selection in rdbms by evolutionary approach. In: Hameurlain A, Liddle S, Schewe KD, Zhou X, editors. Database and expert systems applications. Lecture notes in computer science. Berlin: Springer; 2011. Vol. 6861, p. 475–84. doi:10.1007/978-3-642-23091-2_41.
55. Morelli E, Almeida A, Lifschitz S, Monteiro JM, Machado J. Autonomous re-indexing. In: Proceedings of the 27th annual ACM symposium on applied computing, SAC '12. New York: ACM; 2012. p. 893–7. doi:10.1145/2245276.2245450.
56. Jindal A, Dittrich J. Relax and let the database do the partitioning online. In: Castellanos M, Dayal U, Lehner W, editors. Enabling real-time business intelligence. Lecture notes in business information processing. Berlin: Springer; 2012. vol. 126, p. 65–80. doi:10.1007/978-3-642-33500-6_5.
57. Li L, Gruenwald L. Self-managing online partitioner for databases (smopd): a vertical database partitioning system with a fully automatic online approach. In: Proceedings of the 17th international database engineering and applications symposium, IDEAS '13. New York: ACM; 2013. p. 168–73. doi:10.1145/2513591.2513649.

58. Rösch P, Dannecker L, Färber F, Hackenbroich G. A storage advisor for hybrid-store databases. Proc VLDB Endow. 2012;5(12):1748–58. doi:10.14778/2367502.2367514.
59. Sherkat R, Florendo C, Andrei M, Goel AK, Nica A, Bumbulis P, Schreter I, Radestock G, Bensberg C, Booss D, Gerwens H: Page as you go: piecewise columnar access in sap hana. In: Proceedings of the 2016 international conference on management of data, SIGMOD '16. New York: ACM, 2016. p. 1295–306. doi:10.1145/2882903.2903729.
60. Alagiannis I, Idreos S, Ailamaki A. H_2O: a hands-free adaptive store. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data, SIGMOD '14. New York: ACM, 2014. p. 1103–14. doi:10.1145/2588555.2610502.
61. Arulraj J, Pavlo A, Menon P. Bridging the archipelago between row-stores and column-stores for hybrid workloads. In: Proceedings of the 2016 international conference on management of data, SIGMOD '16. 2016. pp. 583–98 . doi:10.1145/2882903.2915231.
62. Varadarajan R, Bharathan V, Cary A, Dave J, Bodagala S. Dbdesigner: a customizable physical design tool for vertica analytic database. In: 2014 IEEE 30th international conference on data engineering. 2014. p. 1084–95. doi:10.1109/ICDE.2014.6816725.
63. Rasin A, Zdonik S. An automatic physical design tool for clustered column-stores. In: Proceedings of the 16th international conference on extending database technology, EDBT '13. New York: ACM, 2013. p. 203–14. doi:10.1145/2452376.2452402.
64. Mozafari B, Goh EZY, Yoon DY. Cliffguard: a principled framework for finding robust database designs. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data, SIGMOD '15. New York: ACM, 2015. p. 1167–82. doi:10.1145/2723372.2749454.
65. Dageville B, Cruanes T, Zukowski M, Antonov V, Avanes A, Bock J, Claybaugh J, Engovatov D, Hentschel M, Huang J, Lee AW, Motivala A, Munir AQ, Pelley S, Povinec P, Rahn G, Triantafyllis S, Unterbrunner P. The snowflake elastic data warehouse. In: Proceedings of the 2016 international conference on management of data, SIGMOD '16. New York: ACM, 2016. p. 215–26. doi:10.1145/2882903.2903741.
66. Graefe G. Query evaluation techniques for large databases. ACM Comput Surv. 1993;25(2):73–169. doi:10.1145/152610.152611.
67. Abadi DJ, Boncz PA, Harizopoulos S. Column-oriented database systems. Proc VLDB Endow. 2009;2(2):1664–5. doi:10.14778/1687553.1687625.
68. Idreos S, Groffen F, Nes N, Manegold S, Mullender S, Kersten M. MonetDB: two decades of research in column-oriented database architectures. IEEE Data Eng Bull. 2012;35(1):40–5.
69. Google. supersonic library. https://code.google.com/archive/p/supersonic/ (2017). Accessed 12 Feb 2017.
70. Agrawal S, Chu E, Narasayya V. Automatic physical design tuning: workload as a sequence. In: Proceedings of the 2006 ACM SIGMOD international conference on Management of data, SIGMOD '06. New York: ACM, 2006. p. 683–94. doi:10.1145/1142473.1142549.
71. Sockut GH, Iyer BR. Online reorganization of databases. ACM Comput Surv. 2009;41(3):14. doi:10.1145/1541880.1541881.
72. Zukowski M, Héman S, Nes N, Boncz P. Cooperative scans: dynamic bandwidth sharing in a dbms. In: Proceedings of the 33rd international conference on very large data bases, VLDB '07. VLDB Endowment. 2007. p. 723–34.
73. Deshpande A, Ives Z, Raman V. Adaptive query processing. Found Trends Databases. 2007;1(1):1–40. doi:10.1561/1900000001.
74. Abadi D, Myers D, DeWitt D, Madden S. Materialization strategies in a column-oriented dbms. In: IEEE 23rd international conference on data engineering, ICDE. 2007. p. 466–75. doi:10.1109/ICDE.2007.367892.
75. Stonebraker M, Abadi DJ, Batkin A, Chen X, Cherniack M, Ferreira M, Lau E, Lin A, Madden S, O'Neil E, O'Neil P, Rasin A, Tran N, Zdonik S. C-store: a column-oriented dbms. In: Proceedings of the 31st international conference on very large data bases, VLDB '05. VLDB Endowment. 2005. p. 553–64.
76. ONeil PE, ONeil EJ, Chen X. The star schema benchmark (SSB). 2009. http://www.cs.umb.edu/~poneil/StarSchemaB.PDF. Accessed 20 July 2012.
77. Abadi DJ, Madden SR, Hachem N. Column-stores vs. row-stores: how different are they really? In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD '08. New York: ACM, 2008. p. 967–80. doi:10.1145/1376616.1376712
78. Taniar D, Leung CHC, Rahayu W, Goel S. High performance parallel database processing and grid databases. New York: Wiley; 2008.
79. Idreos S, Kersten ML, Manegold S. Self-organizing tuple reconstruction in column-stores. In: Proceedings of the 2009 ACM SIGMOD international conference on management of data, SIGMOD '09. New York: ACM, 2009. p. 297–308. doi:10.1145/1559845.1559878.
80. Graefe G, Kuno H. Self-selecting, self-tuning, incrementally optimized indexes. In: Proceedings of the 13th international conference on extending database technology, EDBT '10. New York: ACM, 2010. p. 371–81. doi:10.1145/1739041.1739087.
81. Idreos S, Kersten ML, Manegold S. Database cracking. In: CIDR. 2007. p. 68–78. http://www.cidrdb.org.
82. Petraki E, Idreos S, Manegold S. Holistic indexing in main-memory column-stores. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data, SIGMOD '15. New York: ACM, 2015. p. 1153–66. doi:10.1145/2723372.2723719.