A Confluence of Column Stores and Search Engines: Opportunities and Challenges

Truls A. Bjørklund
Norwegian University of
Science and Technology
trulsamu@idi.ntnu.no

Johannes Gehrke
Cornell University
johannes@cs.cornell.edu

Øystein Torbjørnsen
Fast Search and Transfer, a
Microsoft® subsidiary
oysteint@microsoft.com

ABSTRACT

IR and DB integration has been a long-withstanding research challenge. Most of the work trying to integrate the two fields is motivated by specific application scenarios. In this paper we approach this problem from another perspective. Instead of focusing on IR and DB as whole fields, we restrict the focus to search engines and column stores. We present observations of similarities in the two technologies, and aggregate information on parallel developments in the two fields. We argue that these developments point towards a confluence of column stores and search engines, and one may in fact argue that this confluence has already started. We evaluate the potential in developing an engine capable of handling the workloads traditionally supported by the different systems, namely decision support and search workloads, by identifying potential opportunities and challenges. The opportunities include potential areas for technology transfer and more efficient support for features. The identified challenges outline areas for future work whose successfulness will help decide whether a confluence of column stores and search engines is feasible.

1. INTRODUCTION

IR and DB integration has been a long-withstanding research challenge. Most of the work on this problem has focused on specific application scenarios requiring systems that are a mixture of traditional database systems and search engines [6, 26, 20]. This paper focuses on the technical similarities between the two fields. Instead of considering database systems as a single field, we focus on a particular class of database systems. Decision Support Systems (DSS) supporting On-Line Analytical Processing (OLAP) is one example of a specialized solution for a particular workload, and we believe that this field has clearer similarities to search engines than other database workloads.

Column-oriented databases (called column stores) have received a lot of attention in recent years through systems like MonetDB [10] and C-store [25]. The primary difference

between such systems and more traditional row-oriented systems is that they store a table one column at a time, instead of one tuple (row) at a time. The primary advantage of column stores is their ability to avoid reading data which is not required to process a query. Column stores also facilitate efficient compression schemes because they compress one column at a time instead of complete tuples. The representation of an attribute value can therefore easily be based on previous values for the same attribute, improving the effectiveness of schemes such as run-length encoding. The cost of updates and of reconstructing tuples during queries are potential disadvantages of column stores [16], but they are considered a good fit for decision support workloads.

Search engines are the primary systems used in information retrieval, known to users through web search engines like Google and Yahoo!. These systems are typically based on an inverted index. An inverted index has for each term, defined as a searchable word in the document collection, an inverted list of the documents containing that term, typically including extra information used to rank the results [31]. If we consider the data stored in a search engine as a large table with terms as attributes and documents as tuples, an inverted index can be interpreted as a column-oriented representation of this table.

The basic observation that search engines and column stores both store tables in a column-oriented fashion motivates the topic of this paper, namely an evaluation of possible synergies obtainable by implementing one system to support both decision support and search workloads. We begin with observations supporting that this confluence has already started in Section 2. Opportunities and challenges associated with an integration are identified in Sections 3 and 4 respectively, before we conclude in Section 5. A summary of the topics discussed in this paper is shown in Table 1.

2. CONFLUENCES

Decision support systems and search engines serve the same purpose in general, namely to provide read-mostly querying capabilities over a knowledge base. Decision support systems have traditionally only supported querying structured data while search engines have focused on collections of unstructured documents. The querying capabilities of the two systems also differ to some extent; decision support systems summarize the answer to queries through aggregations, while search engines answer queries with the top-k documents ranked according to some heuristic. Because the typical query results are so different, it is not obvious

Topic	Aspects	Section
Confluences	Columnar inverted indexes [7]	2.1
	Column stores as search engine back-ends [18, 19]	2.2
	Other hybrid systems [9]	2.3
	Facets vs OLAP dimensions [17, 15, 29]	2.4
Opportunities	Taking advantage of columns	3.1
	Loading	3.2
	Consistency and database offloading	3.3
	Technology transfer of indexing methods	3.4
Challenges	Numbers of columns and sizes	4.1
	Throughput vs. latency	4.2
	Different query languages	4.3

Table 1: Summary of the Paper

whether it is possible to design one system supporting both workloads efficiently. Returning the top-k results has been investigated for structured data as well, but the semantics of ranking documents deviate from ranking tuples [4, 14]. Despite these differences, an analysis of recent developments within both fields suggests that similar ideas are explored. We will take a closer look at some such similarities in this section. An overview of the issues that we discuss is shown in Table 1.

2.1 Columnar Inverted Indexes

As mentioned in the introduction, an inverted index can be interpreted as a column-ordered representation of a table with terms as attributes and documents as tuples, where one inverted list represents one column. The inverted list entries are typically more complex than simple attributes. Exactly what they contain depends on the granularity of the index. In a document-level inverted index, each entry typically consists of the document identifier and the frequency of the term in the document and/or other pre-calculated ranking values. In a word-level inverted index, each entry also contains a list of the actual occurrences of the term in the document [27, 31]. Different types of queries require different subsets of the inverted list entries. Boolean queries only ask for all documents containing some terms combined with boolean operators. Such queries only require reading the document identifiers. Basic traditional ranked queries require reading information used to rank the results, like the frequency of terms in documents. Phrase queries also require reading and processing the occurrences.

A clear advantage of column stores compared to row stores is that, in a column store, data for attributes not required to process a query does not have to be read. As search engine workloads include queries which require reading different subsets of the entries stored in inverted lists, a columnar storage of the inverted lists themselves might also be a good idea. Anh and Moffat experiment with several such schemes in recent work [7], even though the similarities to column stores are not identified. They present three different column-oriented storage schemes for inverted indexes. The first method uses three overall columns, one for document identifiers, one for frequencies and one for the occurrences, and partitions each inverted list into these three columns. The second method stores all the data for each inverted list sequentially, but stores all document identifiers in the list first, before all frequencies and then the occurrence lists. The last approach partitions each inverted list into
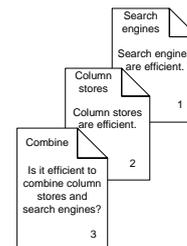

Figure 1: Example documents

Terms		Inverted lists
and	$\longrightarrow$	$(3,1)$
are	$\longrightarrow$	$(1,1)$ $(2,1)$
column	$\longrightarrow$	$(2,2)$ $(3,1)$
combine	$\longrightarrow$	$(3,2)$
efficient	$\longrightarrow$	$(1,1)$ $(2,1)$ $(3,1)$
engines	$\longrightarrow$	$(1,2)$ $(3,1)$
is	$\longrightarrow$	$(3,1)$
it	$\longrightarrow$	$(3,1)$
search	$\longrightarrow$	$(1,2)$ $(3,1)$
stores	$\longrightarrow$	$(2,2)$ $(3,1)$
to	$\longrightarrow$	$(3,1)$

Figure 2: Logical view of inverted index for example documents

blocks and uses column-oriented storage only within each block, a scheme that is similar in spirit to PAX [5]. To see how these index schemes differ from a traditional inverted index, consider the documents in Figure 1. A logical view of a document-level inverted index for these documents is given in Figure 2. We only consider a document-level inverted index here; generalization to a word-level inverted index is straight-forward.

Figure 3 shows how the inverted index for the example documents in Figure 1 would be stored with the different storage schemes. Figure 3a shows a traditional inverted index where all entries in the inverted lists are stored sequentially. In the remaining figures, we assume that each column is stored sequentially, and for simplicity that columns are stored immediately following each other. Under such assumptions, the disk layout for the version with two separate columns is shown in Figure 3b. Figure 3c shows the
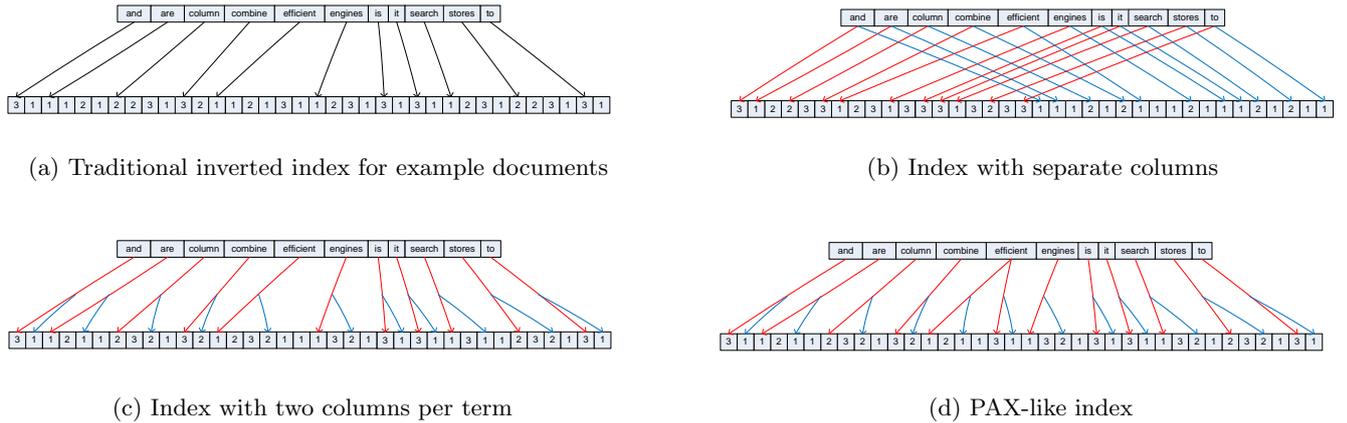
(a) Traditional inverted index for example documents

(b) Index with separate columns

(c) Index with two columns per term

(d) PAX-like index

Figure 3: Inverted indexes with different storage schemes

Term
term ID
term

(a) Term table

Document
document ID
name
length

(b) Document table

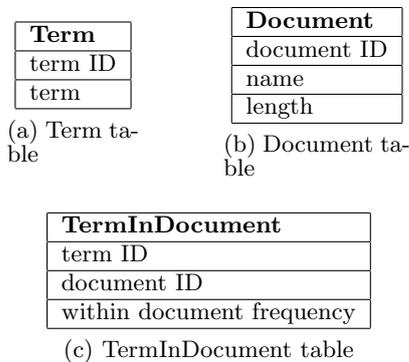| TermInDocument |
| --- |
| term ID |
| document ID |
| within document frequency |

(c) TermInDocument table

Figure 4: Tables used in Search Engine back-end in MonetDB/X100

scheme where there are two columns per term, and Figure 3d shows the scheme where each inverted list is partitioned into blocks. We assume in the figure that each block is capable of storing two inverted list entries, so the only difference from Figure 3c to Figure 3d is in the list for the term "efficient". Apart from the traditional inverted index, all of these schemes are in a sense column-oriented; a fact showing that developments towards a confluence of column stores and search engines have already started in the search engine literature.

2.2 Column Stores as Search Engine Back-Ends

Developments towards a confluence of column stores and search engines have also started in the literature on column stores, where MonetDB/X100 has been suggested as a back-end for a search engine [18, 19]. The data stored in the search engine back-end is organized in standard tables in MonetDB/X100 with the table structure shown in Figure 4.

The **Document** table contains meta-data about the documents that is typically stored in all search engines. A proper storage structure for the **Term** table is essentially similar to the dictionary in an inverted index, which is a searchable structure containing all terms in the document collection. The two last attributes in the **TermInDocument** table contain the data stored in inverted lists with document-level granularity. As MonetDB/X100 is a column-oriented database system, the storage structure resulting from the chosen table layout is actually equivalent to the one tested by Anh and Moffat where all document IDs are stored in one column and all frequencies in another column. The support for compressing one column at a time in column stores enables using similar compression schemes to those used for inverted lists in IR systems, and the resulting index is thus both compact and efficient to decompress [18].

2.3 Other Hybrid Systems

Other examples of systems that start to bridge the gap between column stores and search engines also exist. One example is CompleteSearch, which has support for some database workloads based on the HYB data structure, which is a slightly modified inverted index [9].

2.4 Facets vs. OLAP dimensions

Recall that one difference between search engines and decision support systems is that search engines answer most queries with the top-k documents, while decision support systems typically answer structured queries with aggregations. Through the introduction of faceted search, typical query results in search engines are no longer restricted to ranked lists of documents [17]. Faceted search is an approach that falls between keyword search and browsing, in the sense that it supports both and lets the user choose a strategy on the fly. The result of a query is typically presented as a ranked list of the top-k results along with some facets. These displayed facets represent categories associated with the search, and the results typically include the number of hits within each category from the entire set of results, not only for the top-k. A search involving facets can thus be characterized as processing many group-by's at once in database terminology, one for each displayed facet.

A search on a web page for an electronic store for "phone" could return facets enabling choices between cell phones and regular phones, different brands, and different price ranges. A user could then choose to select one of the top-k results, or to specify the search by adding additional query terms,

or to further specify the query through facets.[1] The ability to drill-down into the search results through facets makes them similar to dimensions in OLAP systems [15, 29].

3. OPPORTUNITIES

Replacing two systems with one which has equal performance and functionality is obviously attractive from a management and cost perspective. Even so, it is probably not enough motivation to go ahead and try to implement such a system. This section outlines other potential advantages that we believe are obtainable in a hybrid column store and search engine system. An overview of the issues that we discuss is shown in Table 1.

3.1 Taking Advantage of Columns

Column stores have the advantage that they avoid reading data that is not required to process a query. They are also able to store sparse data efficiently [1]. In this section we discuss the advantages associated with columnar storage in a search engine, both for the inverted index itself, and when storing meta-data.

3.1.1 Extending the Inverted Index

By vertically partitioning the entries in inverted lists as explained in the previous section, potential disadvantages of adding additional attributes to each entry are limited. If we were to add additional attributes in a traditional inverted index, every query would have to read these attributes, which would obviously slow them down. In the example in the previous section, we constructed a document-level inverted index for the example documents of Figure 1. The advantages of columnar storage are clearer for a word-level index, because it contains the occurrences as well, and the occurrences are not required for simple queries. Examples of even more attributes that can be useful in some queries include attributes to enable personalized search results, and the contexts in which the occurrences are found. The context could for example indicate whether an occurrence is contained in the title or body of the document. A word-level inverted index including such a context for occurrences for Document 1 from Figure 1 is shown in Figure 5.

Terms		Inverted lists
are	$\longrightarrow$	$(1, 1, [4], [body])$
efficient	$\longrightarrow$	$(1, 1, [5], [body])$
engines	$\longrightarrow$	$(1, 2, [1, 3], [title, body])$
search	$\longrightarrow$	$(1, 2, [0, 2], [title, body])$

Figure 5: Logical view of word-level inverted index with context information

The example index in Figure 5 can clearly be extended with more attributes, for instance chapters, sections and pages of occurrences. As we add more attributes to an inverted index, it becomes less and less likely that all queries require all of them. The benefits of column-oriented storage thus increase with more attributes.

3.1.2 Additional Meta-data

As noted in the previous section, all search engines store meta-data about the documents, typically including their

length and name. In faceted search, there are facets associated with each document, and the facets are thus part of the meta-data. To see examples of possible facets, let us consider the documents in Figure 1. We could have one facet describing that a document is about technology. The facet would only be associated with the documents in the collection that have techonological topics, which includes all in our example. We could also include facets that describe the topics of the documents more specifically. A facet describing that a document is about search engines could be associated with documents 1 and 3 in our example, while documents 2 and 3 could be associated with the facet "column stores". Facets with values are also possible, and a simple example of a facet with numerical values is year of publication. This facet would have a value for all documents.

In a large system, the number of different facets is typically large, and most documents have null-values for most of the facets. To store this information, we could leverage that column stores are space efficient for sparse data [1].

3.2 Loading

Loading data has been identified as a key problem in column store databases [2]. C-store was introduced as using LSM-trees, but the solution has never been outlined in detail [25]. Updatable indexes, especially indexes supporting adding more data, have been investigated in detail for search engines on the other hand [11, 12, 13, 23, 21, 22].

While many solutions have been investigated for updatable inverted indexes, solutions based on a hierarchy of indexes tend to perform best overall in experimental comparisons. In such solutions, new documents are added by accumulating an index for them in memory. When the memory set aside for accumulation is filled, the new partial index is merged into the hierarchy. How this merge proceeds depends on the exact method, but in general it is rare that all indexes are merged, a fact that makes the average merge relatively inexpensive. The disadvantage of such approaches is of course that a search for a single term must be performed in several indexes. Retrieving the inverted list for one term with entries in all indexes will thus not take just 1 disk access, as is assumed for a straight-forward inverted index.

The research on loading in search engines seems applicable to column stores, and a hybrid system is a good starting point for exploring this potential technology transfer.

3.3 Consistency and Database Offloading

Although there has been much research on loading data into search engines, there is one fundamental aspect of it that is important for decision support systems which has received limited attention for search engines, namely consistency. In some sense, this difference is a challenge. However, we choose to consider this a potential area for technology transfer because improved support for consistency of operations may enable more extensive use of search engines.

Outlining a particular solution for consistency in a hybrid system is beyond the scope of this paper, but it seems likely that recently developed consistency models with slightly weaker guarantees than the traditional ACID properties can be supported without a major impact on performance. An example of such a model is snapshot isolation, which has been suggested used in C-store [25].

Although not much previous work has been concerned with consistency when adding new documents to a search

---

[1]See http://www.bestbuy.com for an example

engine index, we argue that there are advantages of supporting it, especially if the performance impact is limited. Improved support for consistency in search engines is especially important for database offloading. In database offloading, several related records in a database will typically be transformed to several documents in a search engine, and support for adding these documents as an atomic operation might be important for correctness.

3.4 Technology Transfer of Indexing Methods

Bitmap indexes are commonly used in decision support systems. Traditional bitmap indexes are best fitted for attributes with low cardinality, but bitmap compression techniques have been developed to handle high-cardinality attributes as well [28]. Bitmap indexes have been used previously in information retrieval, but are now mostly replaced by inverted indexes. The primary advantage of inverted indexes which has made them the de facto standard index method in search engines is that they are very compressible [27, 31]. A comparison of the two approaches for decision support workloads might therefore be interesting.

4. CHALLENGES

Even though one can argue that a confluence of column stores and search engines has already started, there are still interesting challenges associated with designing a hybrid system. In particular there are some differences between the workloads traditionally supported by the two systems, and these differences result in different trade-offs. We will outline three of the challenges that have to be solved in order to construct an efficient hybrid system in this section. An overview of the issues that we discuss is shown in Table 1.

4.1 Number of Columns and Column Sizes

Studies have shown that the occurrences of terms in natural language usually follow a Zipfian distribution [30, 8]. In practice, this means that there are a few very common terms that appear in most documents, and that most terms only have one or two occurrences overall. By considering each inverted list to be a representation of a column in the overall term-document table, there are many columns in a search engine index, most of which are very short.

Column stores are mostly used for workloads with limited numbers of columns, but each column is often relatively large. Column stores have been suggested for other application areas than decision support workloads, like storing semantic web data [3]. The solution suggested for semantic web data involves a larger number of columns than what is typically used in a column store, and this is found to cause problems for the directory management in existing column store solutions [24].

Search engine workloads will typically involve more extreme numbers of columns than RDF data, and a hybrid system would thus probably require more sophisticated directory management than column stores have today. The fact that this is a problem for RDF data as well also indicates opportunities. A solution for handling many short columns efficiently in a hybrid system may also help supporting storage of RDF data in column stores more efficiently.

4.2 Throughput Versus Latency

Users expect answers from search engines to be produced in milliseconds even on web-scale document collections. In addition to scaling to many queries per seconds, low latency is thus an important performance requirement for search engines. Query processing with latency-constraints has not been a focus for column stores, as they have rather been optimized for throughput. This difference may lead to quite different trade-offs for different aspects of the system. One example of such an aspect is the unit of transfer from disk. In a system focusing on throughput a large unit of transfer is preferable, because it leads to fewer read operations from secondary storage. Large units of transfer are thus preferable in column stores for decision support workloads. For search engines on the other hand, using large units of transfer may force a query to read much more data from disk than actually required. This problem also becomes more severe when the typical columns are very short, as explained in the last subsection. Even though the data which is unnecessary for one query might be used by another, it will give a higher latency to the query performing the read. A hybrid system must thus be able to balance these two requirements.

The quest for low latency in search engines can even sacrifice correctness in some cases. A simple example of a technique doing just that is removal of stop words [8, 27]. Stop words are the most frequently occurring words in the document collection, and whether such words occur in the query and/or document is not considered vital for the relative ranking of documents. They are thus sometimes removed from the index altogether saving significant amounts of space. Sacrificing correctness is typically not an option in database systems, and a hybrid system must overcome such differences.

Part of the challenge with latency has already gained significance in search engine workloads as well, through the introduction of faceted search. While standard search engine queries only return the top-k results, faceted search queries also present statistics on the complete result set for the shown facets. To do so, a straight-forward solution would have to process the complete result set, posing challenges with respect to latency. To enable efficient query responses in OLAP workloads involving dimensions, it is common practice to materialize certain frequently used data cubes based on selections along the dimensions. Employing such techniques in faceted search, and thus also in a hybrid system, is probably not as beneficial, because the initial selection in search engines is based on keyword queries. Keyword queries create irregular selections compared to selections based on dimensions in a data warehouse.

4.3 Different Query Languages

A potential system based upon the observations in this paper should be able to support both decision support and search workloads. If the challenges outlined in this section are solved, one might be able to support even more workloads efficiently. In particular, such a system would be a good fit for other workloads requiring low latency on read-mostly workloads based on databases and/or search engines, potentially with many columns. One example of such a workload is RDF data, as discussed in Section 4.1, and XML search is another possibility. It is of course attractive in general if the same system can be reused for several workloads, but it also poses problems, like finding a reasonable physical algebra as a basis for the system.

The above mentioned workloads generally support different query languages today. While SQL is standard in struc-

tured data, XPath and XQuery are the most commonly used languages for semi-structured XML data. Search engines on the other hand generally only support queries as lists of words, potentially with a few additional operators like phrases, proximity and some boolean operators. Constructing a hybrid system thus requires finding a reasonable algebra which can form a basis for supporting all the mentioned query languages efficiently, which is clearly a challenge.

It should be noted that through the support of top-k queries for relational data, parts of this problem have already been addressed. But there are semantic differences between ranking documents and tuples [4, 14], making the challenge posed by a hybrid system more involved.

5. CONCLUSION

This paper has discussed parallel developments within search engines and column stores for decision support workloads. These developments point towards a confluence between the two fields. Opportunities and challenges associated with such a confluence have been identified, and Table 1 provides a summary. We believe that addressing the challenges in this paper can have significant practical impact for both search and decision support workloads, in addition to new types of workloads with similar characteristics.

6. REFERENCES

[1] D. J. Abadi. Column stores for wide and sparse data. In *Proc. CIDR*, 2007.

[2] D. J. Abadi. Query execution in column-oriented database systems. MIT PhD Dissertation, 2008.

[3] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach. Scalable semantic web data management using vertical partitioning. In *Proc. VLDB*, 2007.

[4] S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *Proc. ICDE*, 2002.

[5] A. Ailamaki, D. J. DeWitt, M. D. Hill, and M. Skounakis. Weaving relations for cache performance. In *Proc. VLDB*, 2001.

[6] S. Amer-Yahia, P. Case, T. Rölleke, J. Shanmugasundaram, and G. Weikum. Report on the DB/IR panel at SIGMOD 2005. *SIGMOD Record*, 34(4), 2005.

[7] V. N. Anh and A. Moffat. Structured index organizations for high-throughput text querying. In *Proc. SPIRE*, 2006.

[8] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[9] H. Bast and I. Weber. The CompleteSearch engine: Interactive, efficient, and towards IR&DB integration. In *Proc. CIDR*, 2007.

[10] P. A. Boncz, M. Zukowski, and N. Nes. MonetDB/X100: Hyper-Pipelining Query Execution. In *Proc. CIDR*, 2005.

[11] S. Büttcher and C. L. A. Clarke. Indexing time vs. query time: trade-offs in dynamic information retrieval systems. In *Proc. CIKM*, 2005.

[12] S. Büttcher and C. L. A. Clarke. A hybrid approach to index maintenance in dynamic text retrieval systems. In *Proc. ECIR*, 2006.

[13] S. Büttcher, C. L. A. Clarke, and B. Lushman. Hybrid index maintenance for growing text collections. In *Proc. SIGIR*, 2006.

[14] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic ranking of database query results. In *Proc. VLDB*, 2004.

[15] D. Dash, J. Rao, N. Megiddo, A. Ailamaki, and G. Lohman. Dynamic faceted search for discovery-driven analysis. In *Proc. CIKM*, 2008.

[16] S. Harizopoulos, V. Liang, D. J. Abadi, and S. R. Madden. Performance tradeoffs in read-optimized databases. In *Proc. VLDB*, 2006.

[17] M. Hearst, A. Elliott, J. English, R. Sinha, K. Swearingen, and K.-P. Yee. Finding the flow in web site search. *Commun. ACM*, 45(9), 2002.

[18] S. Heman, M. Zukowski, A. P. de Vries, and P. A. Boncz. MonetDB/X100 at the 2006 TREC TeraByte Track. In *Proc. TREC*, 2006.

[19] S. Heman, M. Zukowski, A. P. de Vries, and P. A. Boncz. Efficient and Flexible Information Retrieval Using MonetDB/X100. In *Proc CIDR*, 2007.

[20] N. Kabra, R. Ramakrishnan, and V. Ercegovac. The QUIQ engine: A hybrid IR-DB system. In *Proc. ICDE*, 2003.

[21] N. Lester, A. Moffat, and J. Zobel. Fast on-line index construction by geometric partitioning. In *Proc. CIKM*, 2005.

[22] N. Lester, J. Zobel, and H. Williams. Efficient online index maintenance for contiguous inverted lists. *Inf. Process. Manage.*, 42(4), 2006.

[23] N. Lester, J. Zobel, and H. E. Williams. In-place versus re-build versus re-merge: index maintenance strategies for text retrieval systems. In *Proc. CRPIT*, 2004.

[24] L. Sidirourgos, R. Goncalves, M. Kersten, N. Nes, and S. Manegold. Column-store support for rdf data management: not all swans are white. *Proc. VLDB Endow.*, 1(2), 2008.

[25] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, and S. Zdonik. C-store: a column-oriented dbms. In *Proc. VLDB*, 2005.

[26] G. Weikum. DB&IR: both sides now. In *Proc. SIGMOD*, 2007.

[27] I. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes*. Academic Press, 1999.

[28] K. Wu, E. Otoo, and A. Shoshani. On the performance of bitmap indices for high cardinality attributes. In *Proc. VLDB*, 2004.

[29] P. Wu, Y. Sismanis, and B. Reinwald. Towards

keyword-driven analytical processing. In *Proc. SIGMOD*, 2007.

[30] G. K. Zipf. *Human Behavior and the Principle of Least-Effort*. Addison-Wesley, Cambridge, MA, 1949.

[31] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2):6, 2006.