Scaling up analytical queries with column-stores

Ioannis Alagiannis, Manos Athanassoulis, Anastasia Ailamaki
Ecole Polytechnique Fédérale de Lausanne
Lausanne, VD, Switzerland
{ioannis.alagiannis, manos.athanassoulis, natassa}@epfl.ch

ABSTRACT

As data analytics is used by an increasing number of applications, data analytics engines are required to execute workloads with increased concurrency, i.e., an increasing number of clients submitting queries. Data management systems designed for data analytics - a market dominated by column-stores - however, were initially optimized for single query execution, minimizing its response time. Hence, they do not treat concurrency as a first class citizen.

In this paper, we experiment with one open-source and two commercial column-stores using the TPC-H and SSB benchmarks in a setup with an increasing number of concurrent clients submitting queries, focusing on whether the tested systems can scale up in a single node instance. The tested systems for in-memory workloads scale up, to some degree; however, when the server is saturated they fail to fully exploit the available parallelism. Further, we highlight the unpredictable response times for high concurrency.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems - Concurrency, Query processing, Relational databases

General Terms

Experimentation, Measurement, Performance

Keywords

Concurrency, Column-stores, Data analytics, Scalability

1. INTRODUCTION

In the last few years, data analytics has emerged as a very active research and industrial topic since derived information provides added value both to experimental research and businesses. The insight that data analysis offers to modern businesses, scientists and social applications leads to growing requirements in terms of generating, collecting, managing, and querying data. Today's data analytics engines have to keep up with round-the-clock operation because of the globalization of the economy [4]. Additionally, the nature of analytical queries is expanding from offline analysis to ubiquitous queries. For example, asking from our smartphone the current consumption of minutes and data of our monthly plan initiates a personalized analytical query. Moreover, in recent research it is documented that customers of data analytics products indicate that they will have to routinely execute several hundreds of concurrent queries in the near future [6]. Hence, supporting an increasing number of concurrent analytical queries is a key requirement.

A wealth of systems based on relational data management systems (DBMS) is used to execute analytical queries. While row-stores like IBM DB2 and Oracle have been extensively used for data analytics, column-stores have been proven a very good match for analytical queries since they can efficiently execute queries with small projectivity, aggregates and other specific characteristics often seen in analytical queries. Column-stores were initially developed as research prototypes like MonetDB [5] and C-Store [27] which evolved in commercial products: Vectorwise and Vertica [19] respectively. A variety of commercial column-stores is now available: Infobright [26], SybaseIQ, AsterData, Greenplum and others.

Recent research and optimizations for column-stores focus on single query performance in an attempt to minimize the query response time. This approach is particularly popular in systems that aim primarily at scaling out in a distributed environment. The increase in concurrency requirements described above poses the question how column-stores would perform when scaling up is - at least - equally important as scaling out. This question is further exacerbated by the hardware trends, such as multi-core processors and solid-state storage, offering increased parallelism in processing and accessing data respectively. In this paper we execute an increasing number of concurrent analytical queries as a fundamental test to capture the behavior of column-stores under a heavy load of concurrent queries on memory-resident datasets.

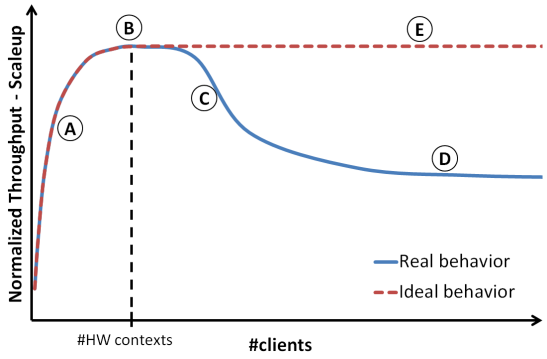


Figure 1: Scaleup of a commercial column-store.

Figure 1 presents the normalized throughput (scaleup) of a commercial column-store based on experiments with a query mix of the TPC-H benchmark. This experiment is conducted on the setup described in Section 4, and measures throughput when running a

DBtest'13, June 24 2013, New York, NY, USA

query mix from an increasing number of clients submitting queries. The y-axis depicts aggregate throughput, showing the scaleup that the system can sustain. We observe that a state-of-the-art commercial column-store cannot sustain peak performance when increasing number of clients, thus suffering from performance degradation in terms of throughput. Section 2 discusses more details about the ideal performance and the implication of increasing concurrency. In the rest of the paper we present different metrics and experiments to understand globally the behavior of column-stores for workloads executed with high concurrency.

Contributions. Our experimental study with three different column-stores (one open-source and two commercial systems) and two benchmarks (TPC-H and SSB) for in-memory datasets makes the following observations:

- Executing queries concurrently leads to linear increase in response time. Even when the server is not saturated, the tested systems fail to fully exploit the available parallelism.
- For large numbers of concurrent queries the systems show large variability in response times. We observe three different cases: (i) System-A shows dual behavior (clients that complete execution early or late), (ii) System-B manages to have balanced progress for most of the outstanding queries leading to relatively lower variation, and (iii) System-C uses admission control to restrain the number of outstanding queries. For all three systems the variation leads to high unpredictability in response times.
- Throughput experiments with a query mix show: (i) when running TPC-H and the server saturates, high concurrency hurts overall performance (32% to 48% decrease depending on the system), and (ii) when running SSB only System-B manages to scale up because it allows concurrent queries to share scans.

2. SCALING UP WITH COLUMN-STORES

Figure 1 shows the real and the ideal behavior of a column-store when an increasing number of clients submit analytical queries. This experiment is conducted with a commercial column-store (described as System-B in Section 4) using a query mix of the TPC-H benchmark. In addition to the discussion of Section 1, in part A of Figure 1 we observe that the throughput of the system scales with the number of clients until point B, at which all hardware contexts are fully utilized and the server is saturated. Ideally, if the database engine has no inefficiencies, for higher number of clients, throughput would plateau at the same level as point B, as the part E of the line shows. The performance of System-B, however, decreases (part C) before it plateaus at a lower rate, about 50% of the peak throughput (part D). There are several messages to be taken from this experiment.

Part A. In the first part of the graph we observe that performance scales up. The actual performance in the saturation point (point B) depends on the database engine used and the exact shape of the curve until point B depends on the capability of the database engine to exploit parallelism for the specific workload. Amdahl's law for scaleup [17] states:

$$scaleup \leq \frac{1}{e + \frac{1-e}{N}},$$

where e is the serial fraction of the database engine, hence $1-e$ the parallel fraction, and N the available hardware parallelism. Using the Karp-Flatt [18] metric we can estimate the serial fraction e of every tested system assuming that we know the level of parallelism p (which is equal to N) and the speedup, ψ, achieved when the maximum level of parallelism is utilized.

Table 1: Serial fraction based on Karp-Flatt metric

Metric	System-A	System-B	System-C
Single-client perf. (Q/h)	304	1683	778
Peak perf. (Q/h)	2575	14567	3693
Speedup	8.47	8.66	4.75
Serial factor	0.09	0.087	0.185

$$e = \frac{\frac{1}{\psi} - \frac{1}{p}}{1 - \frac{1}{p}}.$$

In fact, for the three tested systems running the query mix of the TPC-H benchmark we can infer their serial fraction when executed in our server with 32 hardware contexts ($p = 32$), as shown in Table 1. System-A has 9% serial factor, while System-B has 8.7% and System-C has 18.5%. Notice that the serial factor dictates the relevant scaleup of each system with respect to its single-client performance, hence, it cannot be used for performance comparisons directly. By plugging the serial factor for each system in Amdahl's law for scaleup, and multiplying each curve with the single-thread performance of each system, we predict the ideal performance, as shown in Figure 2.

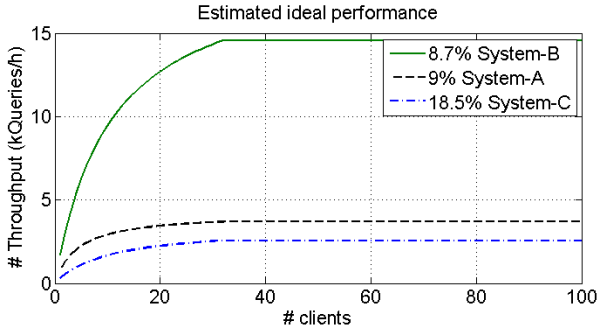


Figure 2: Ideal performance.

Part C. After point B the real behavior of the system differs from the optimal (part E) because the saturation of the processors of the systems allow potential inefficiencies of the engines to kick in and reduce performance. We argue that this performance degradation should be the target of future research on data analytics using column-stores, since the importance of scaling up has increased with new workload and hardware trends.

Part D. While the tested systems cannot sustain maximal performance when the system is saturated we observe that performance plateaus relatively early. This indicates that there is no major bottleneck taking over the execution time, hence, the performance degradation could be addressed with appropriate algorithm redesign, dynamic resource reallocation or admission control aware of the actual CPU utilization. An example of algorithm redesign for analytical queries is the CJOIN [6] operator which departs from the paradigm thread-(or pool-of-threads)-per-query and executes the union of the outstanding queries with a fixed number of threads and distributes the results to the clients in the end.

3. RELATED WORK

This paper focuses on analyzing and presenting the behavior of three column-stores when concurrent analytical queries are submitted by an increasing number of clients. The goal of this analysis is to highlight that column-stores are built with optimizing the response time of a single query as the prime optimization goal, leading to performance degradation and performance unpredictability for high concurrency.

In prior research there is a wealth of techniques to address the concurrency problem: in the optimization phase with *multi-query*

optimization, in the query engine layer with *work sharing*, *materialized views* and *query caching* and in the storage layer with *shared scans*. Orthogonally to these families of techniques, the data reorganization performed by column-stores during query execution (for example by database cracking [15]) generates both reads and writes, but with an appropriate concurrency control mechanism [12] it shows the desired behavior.

Concurrency in the optimization phase. Multi-Query Optimization (MQO) techniques identify common sub-expressions and generate physical plans that share the common computation [10, 23, 24]. Such approaches require queries to be submitted in batches which makes it difficult to respect response time guarantees[1]. Moreover, MQO does not take into account queries that are already in the system - which are potentially long-running. Classic MQO operates on batches of queries only during optimization, and it depends on costly materialization of the common intermediate results.

Concurrency in the query engine. When the executed workload is comprised of a large number of analytical queries over the same data, different queries may share accesses and repeat the same computations. Apart from detecting them in the optimization phase there are techniques to avoid repetition in the execution phase. Any traditional DBMS typically implements both materialized views [22] and query caching [25]. Such techniques help avoiding recomputing identical queries but cannot be predicted by the optimizer nor used in a canonical way. Another option, implemented in MonetDB, is to recycle intermediate results and share them with subsequent queries [16]. Recent research on work sharing, however, offers ad-hoc "collaboration" of the concurrent queries minimizing the overall work done and the number of data accesses. The paradigm of staged databases [13] which is implemented in the QPipe [14] query engine detects common sub-plans during query execution and pipelines the intermediate results avoiding completely re-executing the sub-plan. DataPath [2], SharedDB [11] and the CJOIN [6] operator follow a different approach for work sharing. They create a common global query plan with operators capable of handling tuples of multiple queries and they distribute the tuples in their respective query at the end of the execution phase.

Concurrency in the storage layer. A common technique for the systems using the global query plan approach is the use of shared scans. DataPath [2] uses an array of disks and stores relations in a columnar fashion by hashing and distributing pages to the disks. During query execution, it reads pages from the disks in order, but asynchronously, hence aggregating the throughput of the linear scan of each disk. SharedDB [11] utilizes the Crescando [28] storage manager which uses a circular scan of in-memory table partitions, interleaving reads and writes of a batch of queries. Each Crescando scan applies the updates generated by the current batch of queries for each scanned tuple following the queries' arrival order, and then executes the read requests. CJOIN [6] uses a circular scan as well. Since CJOIN is designed for a star schema (in particular it is evaluated using the Star Schema Benchmark [20]) it is assumed that the dimension tables fit in memory and the fact table is constantly read using a circular scan. Circular scans are implemented in a variety of systems [7, 8, 9, 14] and they can sustain a large number of concurrent tablescans by reducing contention for the buffer pool and the number of accesses to the underlying storage devices. More elaborate techniques for shared scans with coordination are developed for column-stores [29] and for main-memory shared scans [21].

[1]Such guarantees are very important in order to deliver the service-level agreements (SLAs) that a data analytics company typically makes with its clients.

4. METHODOLOGY

In this section, we describe the experimental setup, the workloads employed to study the behavior of the different DBMS, and the applied methodology.

4.1 Experimental Setup

The experiments are conducted in a Sandy Bridge server with a dual socket Intel(R) Xeon(R) CPU E5-2660 (8 cores per socket, 2 logical cores per physical when hyper-threading is on, 32 hardware contexts in total, clocked at 2.20 GHz) equipped with 64 KB L1 cache per core, 256 KB L2 cache per core, 20 MB L3 cache shared, and 128 GB RAM running Red Hat Enterprise Linux 6.3 (Santiago - 64bit) with kernel version 2.6.32. The server is equipped with a RAID-0 of 7 250 GB 7500 RPM SATA disks.

The performed analysis uses two commercial column-stores and a state-of-the-art open-source column-store. To preserve anonymity due to legal restrictions the names of the commercial database systems are not disclosed but instead we refer to them throughout the paper as System-A and System-B respectively. Similarly, the open-source database system is referred as System-C. To enhance query processing performance we have manually generated statistics (for the systems supporting such a feature). All systems are tuned to use all the available hardware contexts from our server and to allow multiple concurrent clients.

4.2 Workloads

We use two benchmarks designed for evaluating data warehouses and decision support systems, the TPC-H decision support benchmark [1] and the Star Schema Benchmark (SSB) [20]. SSB is a modified version of TPC-H in which tables have been altered to represent a star schema and a smaller set of similar queries is used.

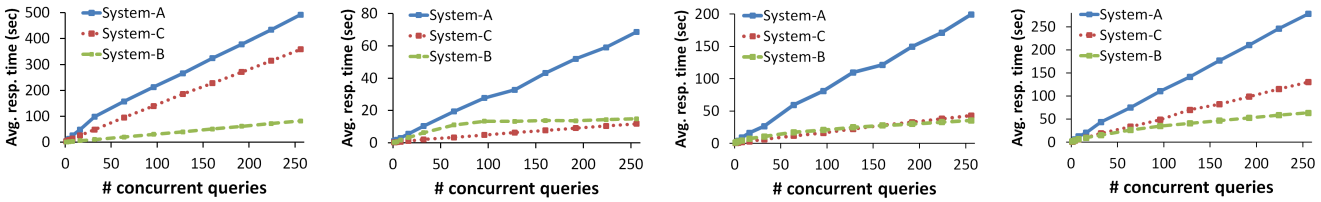
For both benchmarks we perform experiments using databases with scale factor 30 (dataset size 32GB for TPC-H and 18GB for SSB). The TPC-H benchmark typically consists of 22 queries, but we use a subset of 10 queries $Q_{tpch} = \{Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q11, Q12\}$. In the case of SSB, we perform experiments using all the provided queries $Q_{ssb} = \{Q1.1, Q1.2, Q1.3, Q2.1, Q2.2, Q2.3, Q3.1, Q3.2, Q3.3, Q3.4, Q4.1, Q4.2, Q4.3\}$. To enforce variability in our experiments, all the queries are generated based on query templates in which the values of specific predicates vary randomly, though creating queries with similar selectivity. For the TPC-H benchmark the QGen tool is used, while for the SSB a custom SSB-QGen is implemented [3].

To study the throughput as the number of concurrent clients increases in the systems, we generate two types of synthetic workloads W_{tpch} and W_{ssb}. Both workloads contain 25 randomly generated queries selected with the same probability from Q_{tpch} and Q_{ssb} respectively. Each client submits one query from the workload, waits for it to finish and submits the next.

4.3 Design of experiments

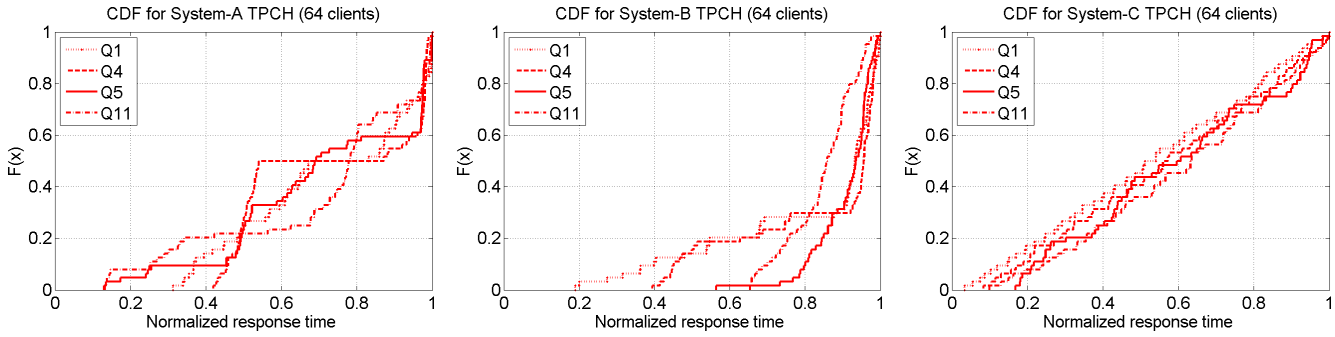
The goal of our experiments is not to compare the systems in terms of performance for individual queries but to examine their behavior as the number of concurrent queries significantly increases. Thus, we vary the number of concurrent queries, the complexity of the queries (e.g. from simple table scans to n-way joins) and the query selectivity space.

For all experiments, the input relations and any intermediate results generated during query processing fit easily in the main memory. We report execution times from hot runs in order to focus on the in-memory query execution part and avoid any interference with I/O operations. In each of the measurements we follow the same sequence of steps. Initially, we warm up the bufferpool of



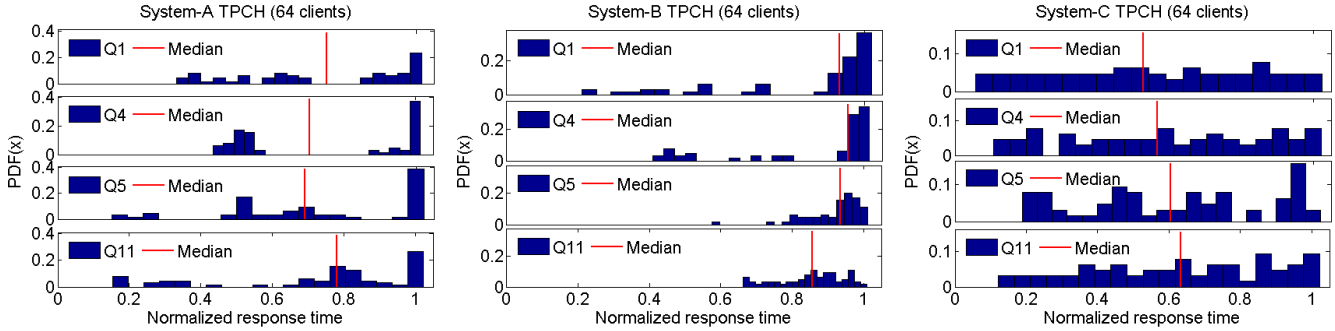
(a) TPC-H - Q1 (b) TPC-H - Q11 (c) SSB - Q2.1 (d) SSB - Q4.1

Figure 3: Average query response time as the number of concurrent queries increases.



(a) (b) (c)

Figure 4: Cumulative distribution function of normalized response time for 64 clients



(a) (b) (c)

Figure 5: Probability distribution function of normalized response time for 64 clients

each system. Then, we initiate multiple connections from clients to the database server and we submit all the queries at the same time for concurrent evaluation. We examine the behavior of all queries in Q_{tpch} and Q_{ssb}, varying the number of concurrent clients from 1 to 256. Due to space restriction, in the following section we report a representative subset of the performed experiments.

5. EXPERIMENTAL ANALYSIS

In this section we evaluate the three column-stores in terms of scalability, predictability of query response and throughput.

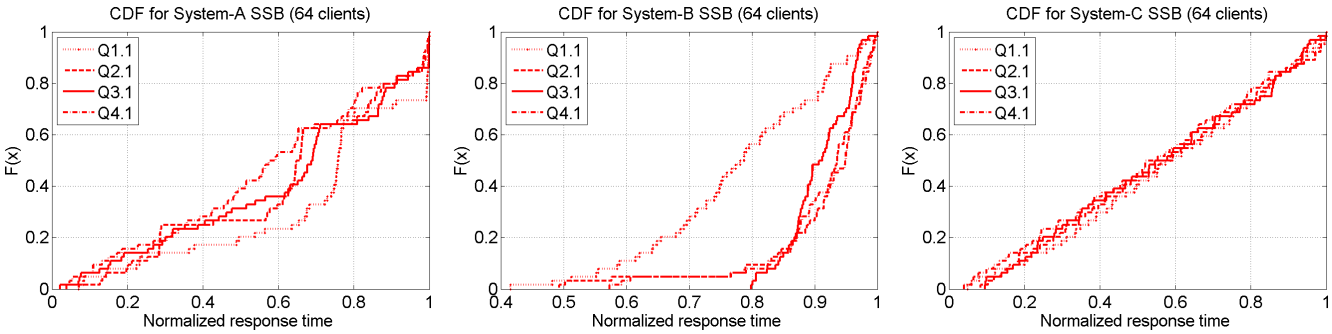
5.1 Scaling up queries

In the following experiment we investigate the behavior of System-A, System-B and System-C as the number of concurrently executing queries increases. We use two memory-resident databases generated from the TPC-H and SSB benchmarks (with scale factor 30). Each client executes one query based on the same query template. The predicates of the input queries have been randomly generated as we describe in Section 4. We present results for two queries of TPC-H ($Q1$ - simple table scan and $Q11$ - 3-way join) and SSB ($Q2.1$ and $Q4.1$ - joins between the fact and a number of dimension tables). Figure 3 plots the average query response time on the y-axis while varying the number of concurrent queries along the x-axis. For these queries, System-B and System-C outperform System-A when it comes to response time. Nevertheless, as the number of concurrent queries we submit increases, we observe a linear increase in response time for all the systems. These results suggest that even for low concurrency the tested DBMS fail to fully exploit the available CPU resources.

Discussion. In an ideal system, we would like the query response time to be independent of the number of queries running concurrently, and the contention for CPU resources to be minimized when the system is saturated.

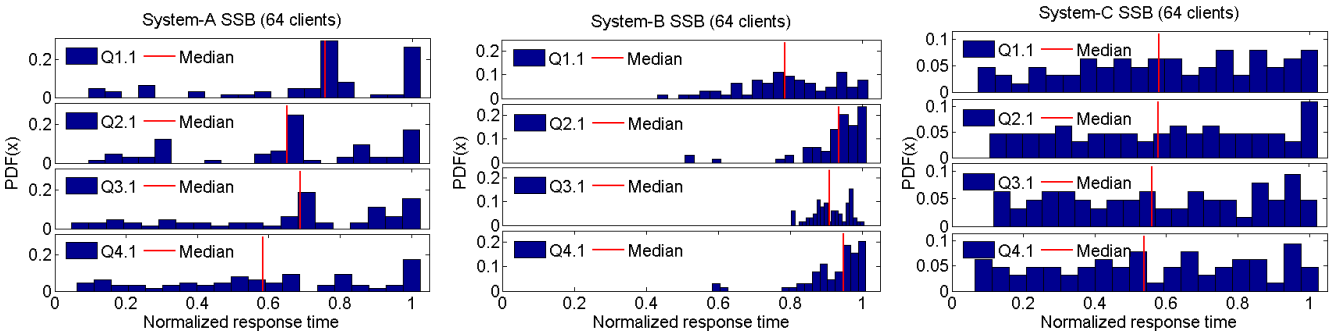
5.2 Variability of query response times

In our next experiment, we demonstrate the predictability of the three DBMS in terms of query response time as multiple queries are submitted. To achieve that, we examine the cumulative distribution function (CDF) of response time for queries with different complexity in TPC-H and SSB. This metric can be used as an indicator of the expected behavior of the tested systems if a new query is submitted. We use the same databases as in the experiment above; however, we additionally include queries $Q4$ and $Q5$ in the case of TPC-H and queries $Q1.1$ and $Q3.1$ in the case of SSB. As be-



(a) (b) (c)

Figure 6: Cumulative distribution function of normalized response time for 64 clients



(a) (b) (c)

Figure 7: Probability distribution function of normalized response time for 64 clients

fore, the queries are randomly generated based on the appropriate query templates of each benchmark. Again, all the clients submit different versions of the same template query concurrently.

The three graphs in Figure 4 show the CDF of normalized response time for System-A, System-B and System-C, while the three graphs in Figure 5 plot the probability distribution function (PDF) along with the median for the same queries. In the above graphs we observe high variability in response times when 64 queries are executed concurrently even though the queries are based on the same template. For example, in System-A the fastest query is computed after 67.9 sec, the slowest after 216.97 sec with an average of 157.67 sec and a median of 180.24 for the 64 clients. On the other hand, System-B not only shows better performance when it comes to response time but a better behavior in terms of variability. The fastest client finishes execution after 4.74 sec while the slowest after 24.9 sec with an average of 20.4 sec and a median of 23.22 sec for the 64 clients. Finally, the response time for $Q1$ in System-C varies from 6.02 sec (fastest) to 180.6 sec (slowest) with an average of 95.24 sec and a median of 94.9.

Overall, we observe different patterns in the strategies for assigning resources to query requests. Figures 4(a) and 5(a) plot the CDF and PDF, respectively, for System-A for 4 TPC-H queries. We observe groups of short, medium and long running queries. For example, in Figure 5(a) response times of $Q4$ are clearly organized into two groups while for $Q11$ we additionally observe a group of short running queries. System-A does not show a specific pattern on how queries are scheduled for execution. System-B shows lower variation among query execution times and follows a more balanced approach when it comes to resource allocation as shown in Figure 5(b). We observe that some queries might complete their execution early but the response time for the majority of queries lays close to the median. On the contrary, System-C shows a different behavior. The observed response times are evenly distributed between the minimum and the maximum. This observation implies that System-C uses an admission control mechanism to restrain the number of queries actually admitted for concurrent execution and limit potential contention allowing for higher throughput.

Figure 6 and Figure 7 summarize the results from the similar experiment with the SSB workload using $Q1.1$, $Q2.1$, $Q3.1$ and $Q4.1$. System-B and System-C show the same behavior as in the case of TPC-H. System-A, similarly, does not show a specific pattern for scheduling query execution and allocating resources among queries.

Discussion. When a single client submits queries in a system, all the available resources can be used to achieve the highest possible performance. Nevertheless, in scenaria where several concurrent queries are expected the database architects should focus on scalability and computing multiple queries within a reasonable time. Submitting a new query to the system should not lead to unpredictable behavior and/or increased response time. Ideally, the query response time should be independent from the current query load in the system and the complexity of the running queries (a similar approach is proposed for row-stores [6]). In practice, a system should provide performance guarantees that each query will eventually be executed in a timely and predictable manner.

5.3 Workload performance analysis

In this experiment, we examine the throughput while varying the number of concurrent clients. We use the same databases as before, and each client executes an instance of W_{tpch} and W_{ssb} presented in Section 4. Figure 8 and Figure 9 plot the throughput on the y-axis (thousand queries per hour) while progressively increasing the number of clients submitting queries concurrently on the x-axis.

For the TPC-H workload (Figure 8), System-B achieves higher throughput in comparison with System-A and System-C, and the peak throughput is achieved for 32 concurrent clients. Regardless of the actual throughput, we observe the same behavior for

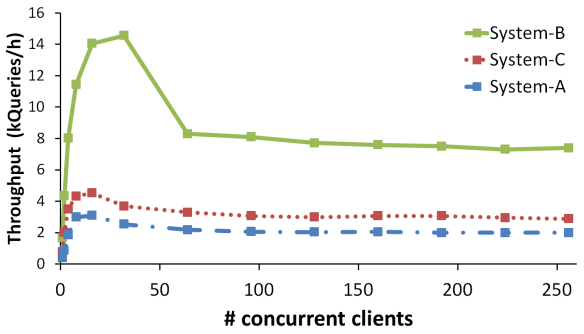


Figure 8: Throughput for TPC-H concurrent queries

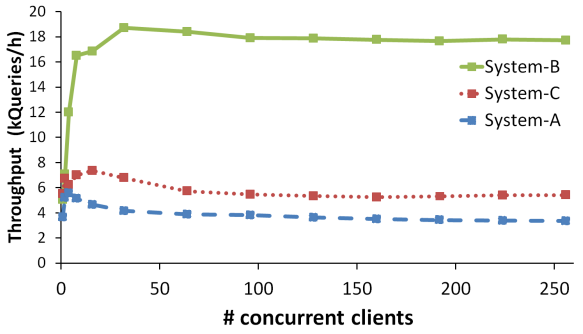


Figure 9: Throughput for SSB concurrent queries

the three systems. Initially, throughput increases but after the saturation point we observe a considerable performance degradation (35% for System-A, 48% for System-B and 32% for System-C). As we explain in Section 2, the performance drop is a consequence of increased resource contention and query engine inefficiencies.

Figure 9 plots the throughput as more clients submit queries for the workload based on the SSB benchmark. Again, System-B significantly outperforms System-A and System-C. In this experiment, System-A achieves the highest throughput for only 4 clients and then throughput decreases up to 39%. On the other hand, System-C achieves the highest throughput for 16 clients and then we observe a degradation in throughput up to 29%. For the SSB workload System-B shows a different behavior. We do not observe any significant drop in throughput. On the contrary, throughput scales up to the saturation point and then remains almost stable regardless of the number of clients. This behavior implies that System-B can exploit data sharing opportunities when multiple queries run concurrently. SSB consists of a big table (lineorder) and 4 small tables. In this case the scan sharing mechanism can efficiently identify data sharing opportunities. On the other hand, TPC-H is more complex (few big tables e.g. lineitem, order, partsupp and more complex queries) which makes finding sharing opportunities harder.

Discussion. In this experiment, apart from the throughput we calculate the CPU utilization as well. We observe that the three tested systems manage to fully utilize the available hardware contexts of our server during their peak performance. Nevertheless, System-A and System-C mainly, and System-B to a lesser extent, fail to maintain this level of performance for the whole duration of the experiment. Ideally, the systems designed for the multicore era should be able to efficiently exploit the massively available parallelism both for static and dynamic workloads.

6. DISCUSSION

The emergence of analytical workloads with high query concurrency as a first class citizen poses the question whether DBMS designed for analytical queries can scale up in addition to scale out. In this paper, we experiment with three column-stores using the TPC-H and SSB benchmarks so as to understand their behavior for high concurrency and highlight opportunities for further optimizations.

A common trend is that when the systems get saturated they cannot sustain stable performance. For increasing concurrency the throughput decreases drastically but the available processors are not always fully utilized. Inefficiencies of the query engines and the lack of aggressive dynamic resource reallocation or of CPU-aware admission control leads to queries underutilizing available resources that cannot exploit idle processors. In recent research on executing analytical workloads with row-stores work sharing is proposed in various forms to enhance performance by locating and exploiting the inherent work sharing opportunities of analytical workloads. While work sharing is no stranger to column-stores, we argue that query evaluation operators in column-stores can be redesigned in order to facilitate ad-hoc and planned sharing in order to use every available opportunity for better scalability when scaling up is the immediate goal.

7. REFERENCES

[1] TPC-H Benchmark: Standard Specification, Revision 2.15.0.

[2] S. Arumugam, A. Dobra, C. M. Jermaine, N. Pansare, and L. Perez. The DataPath system: a data-centric analytic processing engine for large data warehouses. SIGMOD 2010.

[3] M. Athanassoulis and I. Alagiannis. A custom QGen for SSB with randomized parameters. *https://bitbucket.org/manathan/ssb-qgen/overview*, 2013.

[4] M. Athanassoulis, S. Chen, A. Ailamaki, P. B. Gibbons, and R. Stoica. MaSM: efficient online updates in data warehouses. SIGMOD, 2011.

[5] P. Boncz, M. Zukowski, and N. Nes. MonetDB/X100: Hyper-pipelining query execution. *(CIDR)*, 2005.

[6] G. Candea, N. Polyzotis, and R. Vingralek. A scalable, predictable join operator for highly concurrent data warehouses. VLDB, 2009.

[7] L. S. Colby et al. Redbrick Vista: Aggregate Computation and Management. ICDE, 1998.

[8] C. Cook. Database Architecture: The Storage Engine, 2001.

[9] N. Corporation. Teradata Multi-Value Compression V2R5.0, 2002.

[10] S. Finkelstein. Common expression analysis in database applications. SIGMOD, 1982.

[11] G. Giannikis, G. Alonso, and D. Kossmann. SharedDB: killing one thousand queries with one stone. VLDB 2012.

[12] G. Graefe, F. Halim, S. Idreos, H. A. Kuno, and S. Manegold. Concurrency Control for Adaptive Indexing. *PVLDB*, 5(7), 2012.

[13] S. Harizopoulos and A. Ailamaki. A case for staged database systems. CIDR, 2003.

[14] S. Harizopoulos, V. Shkapenyuk, and A. Ailamaki. QPipe: A Simultaneously Pipelined Relational Query Engine. In *SIGMOD*, 2005.

[15] S. Idreos, M. L. Kersten, and S. Manegold. Database Cracking. In *CIDR*, 2007.

[16] M. G. Ivanova, M. L. Kersten, N. J. Nes, and R. A. Gonçalves. An architecture for recycling intermediates in a column-store. In *SIGMOD*, 2009.

[17] R. Johnson, I. Pandis, N. Hardavellas, A. Ailamaki, and B. Falsafi. Shore-MT: a scalable storage manager for the multicore era. In *EDBT*, 2009.

[18] A. H. Karp and H. P. Flatt. Measuring Parallel Processor Performance. *Commun. ACM*, 33(5), 1990.

[19] A. Lamb, M. Fuller, R. Varadarajan, N. Tran, B. Vandier, L. Doshi, and C. Bear. The Vertica Analytic Database: C-Store 7 Years Later . *PVLDB*, 2012.

[20] P. O. Neil, B. O. Neil, and X. Chen. Star Schema Benchmark. 2009.

[21] L. Qiao, V. Raman, F. Reiss, P. J. Haas, and G. M. Lohman. Main-memory scan sharing for multi-core CPUs. *PVLDB*, 2008.

[22] N. Roussopoulos. View indexing in relational databases. *ACM Trans. Database Syst.*, 7, June 1982.

[23] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and Extensible Algorithms for Multi Query Optimization. SIGMOD, 2000.

[24] T. K. Sellis. Multiple-query optimization. *ACM Trans. Database Syst.*, 13, March 1988.

[25] J. Shim, P. Scheuermann, and R. Vingralek. Dynamic caching of query results for decision support systems. SSDBM, 1999.

[26] D. Ślezak and V. Eastwood. Data warehouse technology by Infobright. *SIGMOD*, 2009.

[27] M. Stonebraker et al. C-store: A column-oriented DBMS. *VLDB*, 2005.

[28] P. Unterbrunner, G. Giannikis, G. Alonso, D. Fauser, and D. Kossmann. Predictable performance for unpredictable workloads. VLDB 2009.

[29] M. Zukowski, S. Héman, N. Nes, and P. A. Boncz. Cooperative scans: dynamic bandwidth sharing in a DBMS. VLDB, 2007.