SQALPEL: A database performance platform

DEMO PAPER

M.L. Kersten
CWI
Amsterdam

P. Koutsourakis
MonetDB Solutions
Amsterdam

S. Manegold
CWI
Amsterdam

Y. Zhang
MonetDB Solutions
Amsterdam

ABSTRACT

Despite their popularity, database benchmarks only highlight a small fraction of the capabilities of any given DBMS. They often do not highlight problematic components encountered in real life database applications or provide hints for further research and engineering.

To alleviate this problem we coined *discriminative performance benchmarking* as the way to go. It aids in exploring a larger search space to find performance outliers and their underlying cause. The approach is based on deriving a domain specific language from a sample query to identify and execute a query workload.

The demo illustrates SQALPEL a complete platform to collect, manage and selectively disseminate knowledge, that enables repeatability studies, and economy of scale by sharing performance experiences.

1. INTRODUCTION

Standard benchmarks have long been a focal point in business to aid customers to make an informed decision about a products' expected performance. The Transaction Processing Counsel currently supports several benchmarks ranging from OLTP to IoT. An overview of their active set is shown in Table 1. Surprisingly, the number of publicly accessible results remains extremely low. Just a few vendors go through the rigorous process to obtain results for publication.

In a product development setting, benchmarks provide a yardstick for regression testing. Each new DBMS version or deployment on a new hardware platform ideally shows a better performance. Open-source systems are in that respect not different from commercial products. Performance stability and quality assurance over releases are critically important.

In database research TPC-C and TPC-H are commonly used to illustrate technical innovation. Partly because their description is easy to follow and the data generators easy to deploy.

benchmark	reports	systems reported
TPC-C	368	Oracle, IBM DB2, Sybase, MS SQLserver, SymfoWARE
TPC-DI	0	
TPC-DS	1	Intel
TPC-E	77	MS SQLserver
TPC-H< SF-1000	252	MS SQLserver, Oracle, EXASOL, Action Vector 5.0, Sybase, IBM DB2, Informix, Teradata, Paraccel
TPC-H SF-1000	4	MS SQLserver,
TPC-H SF-3000	6	MS SQLserver, Actian Vector 5.0
TPC-H SF-10000	9	MS SQLserver
TPC-H SF-30000	1	MS SQLserver
TPC-VMS	0	
TPCx-BB	4	Cloudera
TPCx-HCI	0	
TPCx-HS	0	
TPCx-IoT	1	Hbase
see others		

Figure 1: TPC benchmarks

State of affairs. After almost four decades of database performance assessments the state of affairs can be summarized as follows: 1) standard benchmarks are good in underpinning technical innovations, 2) standard benchmarks are hardly representative for real-life workloads[11], and 3) most benchmarking results are kept private unless it is a success story.

SQALPEL addresses 2) using *a new way of looking at performance benchmarking* and 3) by providing a *public repository to collect, administer, and share performance reports.*

Discriminative performance benchmarking. In a nutshell, we believe that database performance evaluation should step away from a predefined query set. For, consider two systems A and B, which may be different altogether or merely two versions of the same system. System B may be considered an overall better system, beating system A on all benchmarked TPC-H queries. This does not mean that no queries can be handled more efficiently in A. These queries might simply not be part of the benchmark suite. Or the improvement is just obtained in the restricted cases covered by the benchmark.

Therefore, the key questions to consider are "what queries perform relatively better on A"? and "what queries run relatively better on B"? Such queries give clues on the side-effects of new features or identify performance cliffs. We coined the term *discriminative benchmark queries* [6] and the goal is to find them for any pair of systems quickly.

Performance repositories. The second hurdle of database

9th Biennial Conference on Innovative Data Systems Research (CIDR '19)
January 13-16, 2019 , Asilomar, California, USA.

benchmarking is the lack of easy reporting and sharing knowledge on experiments conducted. Something akin to a software repository for sharing code. A hurdle to include product results is the "deWitt" clause in most end-user-license agreements, which seemingly prohibits publishing benchmark results for study and research, commentary, and professional advice [1] and which is certainly not intended by the Copyright Laws.[2]

Wouldn't we save an awful lot of energy if experimental data was shared more widely to ground technical innovations? Wouldn't we reduce the cost of running proof-of-concept projects by controlled sharing of experiences within larger teams or the database community at large?

Contributions. In the remainder of this paper we introduce SQALPEL, a SaaS solution to develop and archive performance projects. It steps away from fixed benchmark sets into queries taken from the application and turning them into a grammar as a description of a much larger query space. The system explores this space using a guided random walk to find the discriminative queries. It leads to the following contributions:

- We extend the state of the art in grammar based database performance evaluation.
- We provide a full fledge database performance repository to share information easily.
- We bootstrap the platform with a sizable number of OLAP cases and products.

SQALPEL takes comparative performance assessment to a new level. It is inspired by a tradition in grammar based testing of software [7]. It assumes that the systems being compared understand more-or-less the same SQL dialect and that a large collection of queries can conveniently be described by a grammar. Minor differences in syntax are easily accommodated using *dialect* sections for the lexical tokens in the grammar.

A complete software stack is needed to manage benchmark projects and to deal with sharing potentially confidential information. We are standing on the shoulders of Github in the way they provide such a service. This includes overall organization of projects and their legal structure to make a performance platform valuable to the research community.

Outline. In the remainder of this paper we focus on the design of SQALPEL. In Section 3 we give an overview of the performance benchmark generation. Section 4 addresses access control and legalities. Section 5 summarizes the demo being presented.

2. BACKGROUND

Grammar based testing has a long history in software engineering, in particular in compiler validation, but it also remained a niche in database system testing. In grammar-based testing [7, 12] the predominant approach is to annotate a grammar with probabilistic weights on the productions. It is primarily used to generate test data geared at improved coverage tests for the target system, e.g. compiler [4], or to capture a user interaction with a web-based application. These approaches can be considered static and labor intensive, as they require the test engineer to provide weights and hints up front.

```
query: SELECT ${projection} FROM ${l_tables} $[l_filter] projec
${l_count} ${l_column} ${columnlist}* l_tables: nation columnli
${l_column} l_column: n_nationkey n_name n_regionkey n_comment
l_count: count(*) l_filter: WHERE n_name= 'BRAZIL'
```

Figure 2: Sample SQALPELs **grammar**

Another track pursued is based on genetic algorithms. A good example is the open-source project SQLsmith[3], which provides a tool to generate random SQL queries by directly reading the database schema from the target system. It has been reported to find a series of serious errors, but often using very long runs. Unlike randomized testing and genetic processing we guide the system through the search space by morphing the queries in a stepwise fashion.

Unlike work on compiler technology [8], the grammar based experimentation in the database arena is hindered by the relatively high cost of running a single experiment. Some preliminary work has been focused on generating test data with enhanced context-free grammars [5] or based on user defined constraints in the intermediate results [1]. A seminal work is [10], where massive stochastic testing of several SQL database systems was undertaken to improve their correctness.

A mature recent framework to consider performance analysis of OLTP workloads is described in [3]. It integrates a handful of performance benchmarks and provides visual analytic tools to assess the impact of concurrent workloads. It relies on the JDBC interface to study mostly multi-user interaction with the target systems. It is a pity that the > 150 project forks reported on Github has not yet resulted in a similar number of public accessible reports to increase community insight.

3. SQALPEL PERFORMANCE SPACE

In this section we recap our approach to find discriminative queries based on user-supplied sample queries from their envisioned application setting. We focus on the specification of experiments and running them against target systems. Further details can be found in [6].

3.1 Query space grammar

The key to any SQALPEL performance project is a domain specific language G to specify a query (sub) space. All sentences in the language derived, i.e. $L(G)$, are candidate experiments to be ran against target system(s). The sample grammar in Figure 2 illustrates a query space grammar with seven rules. Each grammar rule is identified by a name and contains a number of alternatives, i.e. free-format sentences with *embedded references* to other rules using an EBNF-like encoding. The SQALPEL syntax is geared at a concise, readable description for the user. Internally, the grammar is normalized by making a clear distinction between rules producing lexical tokens, only governing alternative text snippets, and all others. Furthermore, the validity of the grammar is checked by looking for missing and dead code rules.

[1] https://academia.stackexchange.com/questions/28511/has-the-dewitt-clause-which-prevents-researchers-for-publishing-db-benchmarks-e

[2] https://en.wikipedia.org/wiki/Copyright

[3] https://share.credativ.com/~ase/sqlsmith-talk.pdf

Generation of concrete sentences from the grammar is implemented with a straight-forward recursive descend algorithm. This process stops when the parse tree only contains key words and references to lexical tokens. They will form the *query templates* for a final step, injection of tokens that embody predicates, expressions, and other text snippets.

A naive interpretation of the grammar as a language generator easily leads to an extremely large set of queries. Especially, when a literal token are or when a recursive grammar rule is provided. Therefore, we enforce that the literal tokens are used at most once in a query. This does not rule out that the same literal can be used multiple times. They are differentiated by their line number in the grammar.

Inspired by the observation that most query optimizers normalize expressions lists internally, we can ignore order too in the query generation. It suffices to count the lexical tokens during template generation, e.g. the template `SELECT ${l_column} FROM ${tables}` is derived from the grammar above and can be finalized by choosing a table and column literal list.

We have implemented a full fledged SQL parser that turns a single query, called the baseline query, into a SQALPEL grammar. In real-world applications, a query may be as complex as covering hundres of lines of code. The heuristic applied by the parser is to split the query along projection-list elements, table-expressions, sub-queries, and/or expressions, group-by and order-by terms. The remainders are considered literal tokens.

A good practice for manual construction of a SQALPEL grammar is to gradually increase the complexity of the baseline query. This way the language explosion and subsequent work can be controlled.

3.2 The query pool

In contrast to systems such as RAGS [10] that only randomly generates queries in a brute force manner, we use a query pool. It is populated with the baseline query and some queries constructed from randomly choosen templates. Once a collection has been defined, we can extend the pool by morphing queries based on observed behavior. Three morphing strategies are considered: *alter*, *expand* and a *pruning.*

- *Alter strategy.* We randomly pick a query from the pool and replace a literal. The result is added to the pool unless it was already known.
- *Expand strategy.* We take a query from the pool and search for a template that is slightly larger.
- *Prune strategy.* The reverse operation for expanding a query is to search for a template with slightly fewer lexical classes. It is the preferred method to identify the contribution of sub-queries in highly complex queries.

In [6]the TPC-H benchmark was revisited to assess how large the search space becomes when the SQL queries are converted automatically into a SQALPEL grammar. The number of query templates derived from them vary widely, see Figure 3. This is to be expected, because the grammar produced contains sets of literal classes from which all subsets are considered for template construction. This already results in a combinatorial explosion of templates.

This query pool size is controlled by the project owner. Rules can be fused to reduce the search space by editing the grammar directly. Alternatively, the query pool can be selectively expanded by focusing on a specific strategy and/or a subset of the lexical tokens to guide the project into an area where discriminative queries are expected (See Figure 5). Finally, the number of query templates derived from a grammar is capped.

tag	templates	space	tag	templates	space
Q1	40	9207	Q12	8484	162918
Q2	58160	6354837405	Q13	16	81
Q3	240	29295	Q14	6	21
Q4	28	81	Q15	40	372
Q5	108	96579	Q16	608	25515
Q6	4	15	Q17	26	81
Q7	>100K	–	Q18	576	43659
Q8	480	5478165	Q19	>100K	–
Q9	1512	3528441	Q20	320	3339.0
Q10	384	722925	Q21	18464	4255065
Q11	162	7203	Q22	156	777

Figure 3: TPC-H query space from [6]

3.3 Running experiments

Once a SQALPEL project is defined, people can contribute results using their own DBMS infrastructure using the SQALPEL.PY program. This small Python program contains the logic to call the web-server, requesting a new task and to report back the numbers. It comes with some DBMS client API drivers, but any JDBC enhanced database system can be used directly.

The experiment driver is locally controlled using a configuration file. It specifies the DBMS and host used in the experimental run and the project contributed to. Furthermore, it uses a contributor key supplied separately to identify the source of results later on without disclosing the contributors' identity.

By default each experiment is ran five times and the wall clock time for each step is reported. When available, the system load at the beginning and end of the experimental run is kept around. It is the sole OS environment attribute easily accessible in a Linux environment. Its 1,5, and 10 minute CPU load averages provide an indication of processor load during the runs. An open-ended key-value list structure can be returned to keep system specific performance indicators for post inspection.

4. SHARING EXPERIENCES

In this section we present an overview of the SQALPEL architecture to define and manage performance projects. It addresses the second issue, i.e. improve sharing of database performance information. A problem that calls for both legal and technical solutions.

4.1 Design considerations

A major shortcoming of TPC benchmark reports is their lack of product coverage. Therefore, the design starting point for SQALPEL is to recognize that performance data only makes sense if you can easily document it and share it. This immediately takes TPC benchmarking outside the realm of where it is now, i.e. a by-product to finalize a scientific paper to proof a point. Every database researcher has to partly re-run them on his novel system and compare it with an open-source DBMS. To circumvent a holistic comparison with mature products, small-scale micro-benchmarks further aid in the analysis of specific system functions or scientific experiments.

Since public benchmarks rarely reflect the application specifics we are also faced with another dilemma. Should we disclose the database schema, sample data, and the queries of our intended product with others? Probably not. However, we can greatly benefit from the insights of others who provide reports on generic or abstracted case studies that are close to our intended application. Rather then making our context public we should establish collaborative teams.

4.2 Access control in SQALPEL

This leads to a clear distinction of the kind of projects to facilitate, i.e. public and/or private ones, with access control over who can read/ modify/ comment. Much like how projects are managed on Github. A performance project is initiated and owned by someone, the *project leader*, who acts as a moderator for quality assurance. He invites or accepts *members* for providing insights or to run the experiments in their own DBMS context.

A project declared public may not contain references to private DBMS and host settings. Otherwise, all information is available in read-only mode. The project owner is the moderator who can actively expand the query pool and manage the results. Registered users can leave comments behind to improve upon the presentation, highlight issues, or suggest other experiments.

Contrarily, a private project enables users to experiment with proprietary database instances or products. For contributors the information shielding is lifted. They can contribute results and inspect all results of the projects they are members of. There is no upper limit on the number of contributors per project.

4.3 Legalities

A long standing complication is that many of the software licenses are seemingly extremely restrictive when it comes to sharing experiences on their use. To quote a typical clause in a End-User-License of a commercial product: *You must obtain Microsoft's prior written approval to disclose to a third party the results of any benchmark test of the software.*

It is time to throw away the fear to publish experiences with commercial products. This does not mean one should bash a product, because this could lead to a lawsuit on damaging the vendors' credentials. Instead, the sharing should be based on scientific rigor and aimed towards innovation [9]. In particular, the documentation of the DBMS knobs, the database schema meta data, e.g. index use, partitioning, compression,.. the data distribution, just to name a few.

Although users are free to bind themselves in such limiting contracts, the repercussions on violation of the "deWitt" clause has, to my knowledge, not been tested extensively in court.[4] **Furthermore, the Copyright Laws (in the US) permit "fair use" for research: "In many jurisdictions, copyright law makes exceptions to these restrictions when the work is copied for the purpose of commentary or other related uses."** [5]

<Panos>: I propose we change the bold text in the two paragraphs above with the following: "We believe that these restrictions are fundamentally incompatible with the essence of the scientific process. This is a discussion that needs to take place as a matter of some urgency in scientific, legal and commercial fora. One of the goals of this paper is to open such a discussion."

We believe that the open-source legal framework of Github [6], backed by the SQALPEL access control policies, strikes a balance between the need for public and confidential information. If a vendor believes a project infringes on their license, they can follow the *notice and take down* method, which is a process operated by an online host in response to court orders or allegations that content is illegal. Content is removed by the host following notice.

5. DEMO SCENARIO

SQALPEL is a SaaS prototype under active development [7]. In this demo we report on functionality and features mostly. The purpose is definitely not to engage into a deep performance analysis of a specific target DBMS, but on assessment of the SQALPEL design and heuristics itself to steer us into the right direction.

At the time of writing it supports all JDBC-based database systems, contains sample projects inspired by TPC-H, SSBM, airtraffic, and platforms ranging from a Raspberry Pi up to Intel Xeon E5-4657L servers with 1TB RAM.

5.1 The software platform

SQALPEL is built as a web-based software platform for developing, managing, and sharing experimental results. The GUI is built around Python and the Flask and Bokeh libraries, which already provide a set of visual data analytics functions. The server is deployed in the cloud for ease of access. The system comprises around 12K lines of Python code, 70 Jinja2 templates, and 6K lines of HTML. Pages are generated at the server side, including the necessary JavaScript for interactive manipulation. The session state is kept within the server and is considered global within a browser.

5.2 Top menu navigation

The top level menu is kept simple. A straightforward user administration is provided based on an unique nickname and a valid email to reach out to its owner. Email addresses are never exposed unless legal interaction is required.

The global DBMS catalog describes all database systems considered and the platform catalog provides an overview of the hardware platforms deployed. Both can be readily extended to include information about new platforms used, or the DBMS configuration parameters for experimentations.

5.3 Project experiments

Once the anticipated DBMS and host are known, either an existing project is opened or a new one started. Its synopsis contains all information to repeat the experiments, provides proper attribution to the database generator developers, and marks the project as either public/ private.

[4] https://academia.stackexchange.com/questions/28511/has-the-dewitt-clause-which-prevents-researchers-for-publishing-db-benchmarks-e, http://www.itprotoday.com/microsoft-sql-server/devils-dewitt-clause

[5] https://en.wikipedia.org/wiki/Copyright

[6] https://github.com/github/site-policy/

[7] http://www.sqalpel.com

An experiment consists of a sample SQL query converted into a scalpel grammar (see Figure 4). In case the grammar produces too many semantic incorrect queries or leads to exorbitant large space, a manual edit of the grammar is called for. Some alternatives can be removed by making e.g. join-paths explicit.

5.4 Query pool

With the scalpel grammar in place we can switch to the query pool (Figure 5). It is initialized with the baseline query. The pool is extended using the strategies described in Section 3.2. Most likely you start with a number of random queries and expand as you learn more about the impact on the target systems. A fine grained control is provided by making explicit what lexical terms should/shouldn't be included in the queries being generated. It helps in avoiding experiments where the performance impact is already covered by others, e.g. predicates with comparable impact are removed.

5.5 Running experiments

With a pool of queries in place, we can invite contributors to provide actual performance data. They use the SQL-PROBE experiment driver. Its basic interaction is to call the SQALPEL webserver for a task from a project/ experiment pool, execute it, and report the findings. The contributor is in full control over the target platform and only data is shared that aids in the analysis. For fair comparison of results, the best practices of experimentation should be followed. Documentation of the settings used in the target system is critical for a proper interpretation. In particular DBMS server settings may have an effect on the performance, e.g. addition of indices without reporting them would confuse others to interpret the results.

Each query is ran against a single DBMS + host combination. Its status is kept around in a queue, which enables killing queries that got stuck or when the results of an experiment are not delivered in reasonable time.

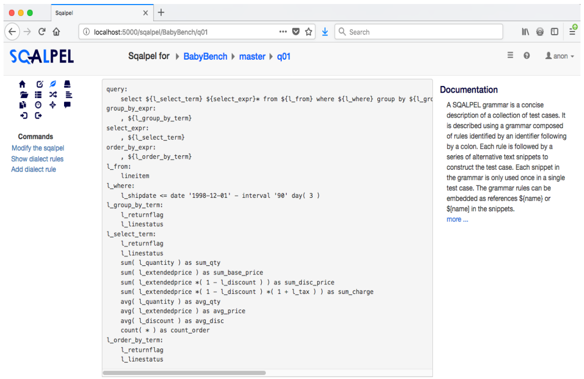
All raw results are collected in a results table for close inspection. One particular use case is to remove results from target systems that require a re-run, e.g. they are measured incorrectly. It is often better to hide these suspicious results until clarification has been obtained from the contributor.

5.6 Visual analytics

Once the results come back from the contributors, they can be analysed with a few built-in visual analytics commands or exported in CSV for post-processing. Figure 6 shows the execution time of queries in a single experiment. The dashed lines illustrate the morphing action taken. The color coding for {alter, expand, prune} morphing is {purple, green, blue}. Queries that result in an error are shown as gray dots. Note that they can be morphed into valid queries later on. The node size illustrates the number of components in the query. Hovering over a node shows the details of the run.

A principle component analysis of the lexical terms in the queries may indicate costly ones (see Figure 7). The dominant term in Q1 for a MonetDB installation is:

```
sum(l_extendedprice*(1 - l_discount)*(1 + l_tax)) as sum_charge
```

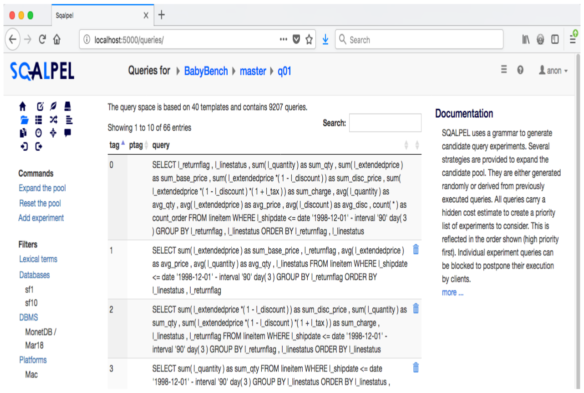


```
query:
    select ${l_select_term} ${select_expr}* from ${l_from} where ${l_where} group by ${l_gro
group_by_expr:
    , ${l_group_by_term}
select_expr:
    , ${l_select_term}
order_by_expr:
    , ${l_order_by_term}
l_from:
    lineitem
l_where:
    l_shipdate <= date '1998-12-01' - interval '90' day( 3 )
l_group_by_term:
    l_returnflag
    l_linestatus
l_select_term:
    l_returnflag
    l_linestatus
    sum( l_quantity ) as sum_qty
    sum( l_extendedprice ) as sum_base_price
    sum( l_extendedprice *( 1 - l_discount ) ) as sum_disc_price
    sum( l_extendedprice *( 1 - l_discount ) *( 1 + l_tax ) ) as sum_charge
    avg( l_quantity ) as avg_qty
    avg( l_extendedprice ) as avg_price
    avg( l_discount ) as avg_disc
    count( * ) as count_order
l_order_by_term:
    l_returnflag
    l_linestatus
```

Documentation

A SQALPEL grammar is a concise description of a collection of test cases. It is described using a grammar composed of rules identified by an identifier following by a colon. Each rule is followed by a series of alternative text snippets to construct the test case. Each snippet in the grammar is only used once in a single test case. The grammar rules can be embedded as references ${name} or $[name] in the snippets.
more ...

Figure 4: Query scalpel



The query space is based on 40 templates and contains 9207 queries.

Showing 1 to 10 of 66 entries

tag	ptag	query
0		SELECT l_returnflag , l_linestatus , sum(l_quantity) as sum_qty , sum(l_extendedprice) as sum_base_price , sum(l_extendedprice *(1 - l_discount)) as sum_disc_price , sum(l_extendedprice *(1 - l_discount) *(1 + l_tax)) as sum_charge , avg(l_quantity) as avg_qty , avg(l_extendedprice) as avg_price , avg(l_discount) as avg_disc , count(*) as count_order FROM lineitem WHERE l_shipdate <= date '1998-12-01' - interval '90' day(3) GROUP BY l_returnflag , l_linestatus ORDER BY l_returnflag , l_linestatus
1		SELECT sum(l_extendedprice) as sum_base_price , l_returnflag , avg(l_extendedprice) as avg_price , avg(l_quantity) as avg_qty , l_linestatus FROM lineitem WHERE l_shipdate <= date '1998-12-01' - interval '90' day(3) GROUP BY l_returnflag ORDER BY l_linestatus , l_returnflag
2		SELECT sum(l_extendedprice *(1 - l_discount)) as sum_disc_price , sum(l_quantity) as sum_qty , sum(l_extendedprice *(1 - l_discount) *(1 + l_tax)) as sum_charge , l_linestatus , l_returnflag FROM lineitem WHERE l_shipdate <= date '1998-12-01' - interval '90' day(3) GROUP BY l_returnflag , l_linestatus ORDER BY l_linestatus
3		SELECT sum(l_quantity) as sum_qty FROM lineitem WHERE l_shipdate <= date '1998-12-01' - interval '90' day(3) GROUP BY l_linestatus ORDER BY l_linestatus ,

Documentation

SQALPEL uses a grammar to generate candidate query experiments. Several strategies are provided to expand the candidate pool. They are either generated randomly or derived from previously executed queries. All queries carry a hidden cost estimate to create a priority list of experiments to consider. This is reflected in the order shown (high priority first). Individual experiment queries can be blocked to postpone their execution by clients.
more ...

Figure 5: Query pool

It is by far the most expensive component. The underlying reason stems from the way MonetDB evaluates such expressions, which include type casts to guard against overflow and creation of fully materialized intermediates.

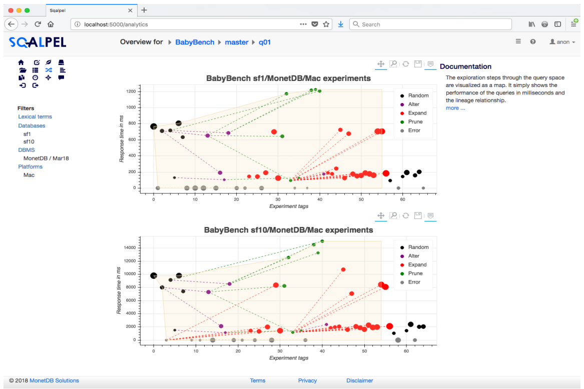
Relative speedup between different versions of a system can be directly visualized (Figure 8). It should not come as a surprise that speedup factors of an individual query only tells part of the story. The figure shows that the base line query SF 1 Q1 runs about a factor 8 slower on a 10 times larger database instance. However, looking at the query variations it actually shows a spread of a factor 8-14. The outliers are of particular interest to dig into. The next step would be to determine their differences. For this we use a differential page (Figure 9). It highlights the differences in query formulation and gives an overview of the performance on various systems. This provide valuable insights to focus experimentation and engineering.

6. SUMMARY AND CONCLUSIONS

We provided a progress report on the SQALPEL project, a sharp tool in the hands of system architects, DBAs and end-users to facilitate sharing experimental data. Saving an enormous amount of energy for the (research) users when confronted with best-of-breed product selection, or best-of-breed functionality offerings, or simply measuring progress in science by standing on the shoulders of giants.

Acknowledgments

This research has received funding from the European Union's Horizon 2020 research and innovation programme under Grant

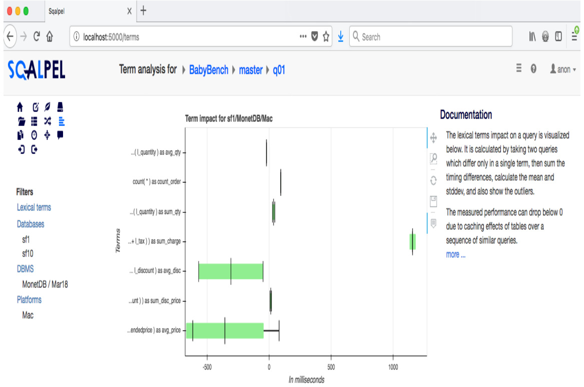


Overview for ▸ BabyBench ▸ master ▸ q01

Documentation

The exploration steps through the query space are visualized as a map. It simply shows the performance of the queries in milliseconds and the lineage relationship.
more ...

Figure 6: Experiment history


Sqalpel
localhost:5000/terms
Search
SQALPEL
anon
Filters
Lexical terms
Databases
sf1
sf10
DBMS
MonetDB / Mar18
Platforms
Mac
Term impact for sf1/MonetDB/Mac
Terms
...(l_quantity) as avg_qty
count(*) as count_order
...(l_quantity) as sum_qty
...+ l_tax)) as sum_charge
... l_discount) as avg_disc
...unt)) as sum_disc_price
...endedprice) as avg_price
-500
0
500
1000
In milliseconds

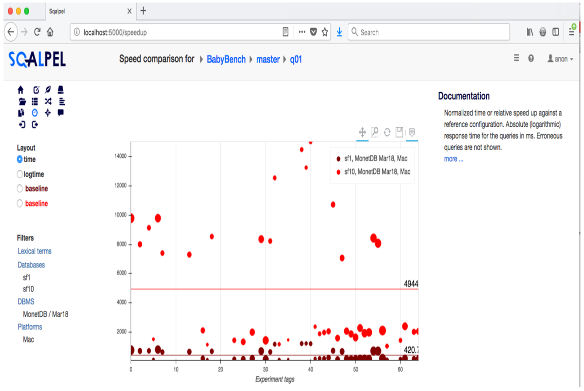
Term analysis for ▸ BabyBench ▸ master ▸ q01

Documentation

The lexical terms impact on a query is visualized below. It is calculated by taking two queries which differ only in a single term, then sum the timing differences, calculate the mean and stddev, and also show the outliers.

The measured performance can drop below 0 due to caching effects of tables over a sequence of similar queries.
more ...

Figure 7: Principle components

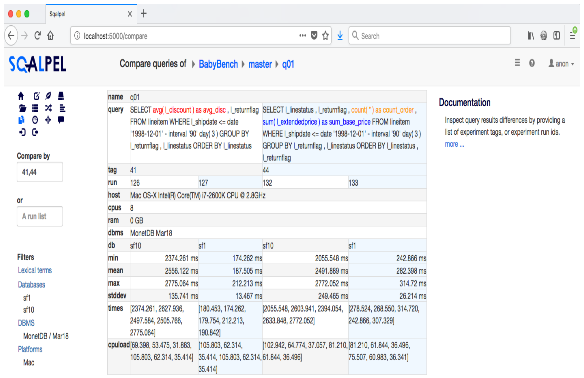


Speed comparison for ▸ BabyBench ▸ master ▸ q01

Documentation

Normalized time or relative speed up against a reference configuration. Absolute (logarithmic) response time for the queries in ms. Erroneous queries are not shown.
more ...

Figure 8: Query speedup



Compare queries of ▸ BabyBench ▸ master ▸ q01

Compare by

41,44

or

A run list

name	q01			
query	SELECT avg(l_discount) as avg_disc , l_returnflag FROM lineitem WHERE l_shipdate <= date '1998-12-01' - interval '90' day(3) GROUP BY l_returnflag , l_linestatus ORDER BY l_linestatus		SELECT l_linestatus , l_returnflag , count(*) as count_order , sum(l_extendedprice) as sum_base_price FROM lineitem WHERE l_shipdate <= date '1998-12-01' - interval '90' day(3) GROUP BY l_returnflag , l_linestatus ORDER BY l_linestatus , l_returnflag	
tag	41		44	
run	126	127	132	133
host	Mac OS-X Intel(R) Core(TM) i7-2600K CPU @ 2.8GHz			
cpus	8			
ram	0 GB			
dbms	MonetDB Mar18			
db	sf10	sf1	sf10	sf1
min	2374.261 ms	174.262 ms	2055.548 ms	242.866 ms
mean	2556.122 ms	187.505 ms	2491.889 ms	282.398 ms
max	2775.064 ms	212.213 ms	2772.052 ms	314.72 ms
stddev	135.741 ms	13.467 ms	249.465 ms	26.214 ms
times	[2374.261, 2627.936, 2497.584, 2505.766, 2775.064]	[180.453, 174.262, 179.754, 212.213, 190.842]	[2055.548, 2603.941, 2394.054, 2633.848, 2772.052]	[278.524, 268.550, 314.720, 242.866, 307.329]
cpuload	[69.398, 53.475, 31.883, 105.803, 62.314, 35.414]	[105.803, 62.314, 35.414, 105.803, 62.314, 35.414]	[102.942, 64.774, 37.057, 81.210, 61.844, 36.496]	[81.210, 61.844, 36.496, 75.507, 60.983, 36.341]

Documentation

Inspect query results differences by providing a list of experiment tags, or experiment run ids.
more ...

Figure 9: Query differentials

Agreement no. 732366 (ACTiCLOUD).

7. REFERENCES

[1] Carsten Binnig, Donald Kossmann, Eric Lo, and M. Tamer Özsu. QAGen: Generating query-aware test databases. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 341–352, New York, NY, USA, 2007. ACM.

[2] Alexander Böhm and Tilmann Rabl, editors. *Proceedings of the 7th International Workshop on Testing Database Systems, DBTest@SIGMOD 2018, Houston, TX, USA, June 15, 2018*. ACM, 2018.

[3] Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudré-Mauroux. Oltp-bench: An extensible testbed for benchmarking relational databases. *PVLDB*, 7(4):277–288, 2013.

[4] Hai-Feng Guo and Zongyan Qiu. Automatic grammar-based test generation. In Hüsnü Yenigün, Cemal Yilmaz, and Andreas Ulrich, editors, *Testing Software and Systems*, pages 17–32, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[5] Johannes Härtel, Lukas Härtel, and Ralf Lämmel. Test-data generation for xtext. In Benoît Combemale, David J. Pearce, Olivier Barais, and Jurgen J. Vinju, editors, *Software Language Engineering*, pages 342–351, Cham, 2014. Springer International Publishing.

[6] Martin L. Kersten, Panagiotis Koutsourakis, and Ying Zhang. Finding the pitfalls in query performance. In Böhm and Rabl [2], pages 3:1–3:6.

[7] Ralf Lämmel and Wolfram Schulte. Controllable combinatorial coverage in grammar-based testing. In M. Ümit Uyar, Ali Y. Duale, and Mariusz A. Fecko, editors, *Testing of Communicating Systems*, pages 19–38, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[8] William M. McKeeman. Differential testing for software. *DIGITAL TECHNICAL JOURNAL*, 10(1):100–107, 1998.

[9] Mark Raasveldt, Pedro Holanda, Tim Gubner, and Hannes Mühleisen. Fair benchmarking considered difficult: Common pitfalls in database performance testing. In Böhm and Rabl [2], pages 2:1–2:6.

[10] Donald R. Slutz. Massive stochastic testing of SQL. In Ashish Gupta, Oded Shmueli, and Jennifer Widom, editors, *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 618–622. Morgan Kaufmann, 1998.

[11] Adrian Vogelsgesang, Michael Haubenschild, Jan Finis, Alfons Kemper, Viktor Leis, Tobias Mühlbauer, Thomas Neumann, and Manuel Then. Get real: How benchmarks fail to represent the real world. In Böhm and Rabl [2], pages 1:1–1:6.

[12] Kamal Z. Zamli, Mohammad F.J. Klaib, Mohammed I. Younis, Nor Ashidi Mat Isa, and Rusli Abdullah. Design and implementation of a t-way test data generation strategy with automated execution tool support. *Information Sciences*, 181(9):1741 – 1758, 2011.