LDBC

Cooperative Project

FP7 – 317548

D1.1.2 Benchmark principles and methods

Coordinator: [Renzo Angles (VUA)]

With contributions from: [Peter Boncz (VUA), Josep Lluís Larriba (UPC), Norbert Martínez (UPC), Barry Bishop (ONTO)]

1st Quality Reviewer: Orri Erling (OGL)
2nd Quality Reviewer: Thomas Neumann (TUM)

Deliverable nature:	Report (R)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	M12
Actual delivery date:	M12
Version:	1.0
Total number of pages:	47
Keywords:	benchmarking, benchmark development process, choke points

Abstract

The objective of this deliverable is to present an initial set of principles and methods for defining, performing, auditing and publishing RDF and graph database benchmarks in LDBC.
We start the deliverable with a review of the process models defined by influential organizations dedicated to benchmarking and standardization (i.e. TPC, SPEC, and W3C). In our study, we put special interest in two main activities: the development of benchmarks, and the execution of benchmarks. Based on this study, we present a methodology for benchmark development which is guided by the identification and modeling of technological challenges called "choke points". Additionally we provide guidelines concerning performance metrics, benchmark execution rules, and benchmarking results evaluation (i.e. auditing).

Executive Summary

Benchmarking is a systematic and continuous process of creation, measurement, comparison and improvement. The process starts with the creation of a benchmark, which usually results in a collection of software tools and a specification with clear rules for executing the benchmark. Once the benchmark has been created, it can be used to measure the performance of the target systems. The results of executing the benchmark can be used, by vendors and consumers (systems users), to compare the performance of the evaluated systems. Finally, the comparison results cause improvements: vendors improve their technology in order to be competitive, and current benchmark design can be improved to cover new necessities and application domains.

In the previous deliverable D1.1.1, we presented the state of the art of benchmarks for core database functionalities, graph databases, RDF databases, and RDF data integration approaches. Additionally, we discussed benchmarking principles focusing mainly on the purpose and scope of benchmarks. In this deliverable, we review the process models defined by influential organizations dedicated to benchmarking and standardization. The objective is to present an initial set of principles and methods for defining, performing, auditing and publishing RDF and graph database benchmarks in LDBC.

We start our study by analyzing organizations related to benchmarking, specifically the Transaction Processing Performance Council (TPC) and the Systems Performance Evaluation Cooperative (SPEC). In our study, we put special interest in two main processes inside such organizations: (1) the benchmark development process; and (2) the benchmark execution process. Additionally, we review the process defined by the World Wide Web Consortium (W3C) for the creation of Web standards. It is particularly interesting to understand the process model behind the creation of standards in an open and community oriented organization like the W3C.

The *benchmark development process* is a series of management and methodological activities whereby a group of people creates a benchmark. By management activities we mean the organizational protocols (i.e. established rules) to control the process of benchmark development, and by methodological activities we mean the principles, methods and steps for creating a benchmark. For example, the TPC defines its "Benchmark Development Cycle" as a 9-step process starting with the submittal of a benchmark proposal and finishing with the creation of a maintenance subcommittee for the created benchmark. A similar description is presented for SPEC. Such descriptions could be very useful to define the management activities in LDBC. Unfortunately, there is not much information about methodological activities in such organizations.

Although several benchmarks for databases have been proposed, there is not a standard recipe behind their development. In this sense, a *methodology for benchmark development* is fundamental as a formal definition of all the elements that participate in the process of benchmark development and the guidelines supporting such process. In this deliverable we present a methodology for benchmark development by describing the following elements: *roles and bodies*, concerning the persons (individual or grouped) participating of the benchmark creation; *design principles*, concerning the fundamental rules that directs the development of a benchmark (e.g. relevance and simplicity); and *development process*, concerning a series of steps to develop a benchmark (i.e. conceptualization, analysis, design, implementation, testing, and distribution). The proposed development process is based on the definition and modeling of technical challenges called "choke points'. This approach depends on a combination of workload input by end users, and access to true technical experts in the architecture of the systems being benchmarked. The overall goal of the choke-point based approach is to ensure that a benchmark covers a spectrum of technical challenges, forcing systems onto a path of technological innovation in order to score good results.

The *benchmark execution process* considers the application of a benchmark over a database system, called the System Under Test (SUT). This process includes the following steps: benchmark implementation, that consists in the configuration and tuning of the benchmark and the SUT; performance measurement, that implies running the benchmark over the SUT and report the measurement results; benchmarking results validation, that consists in reviewing the results and verify compliance with the benchmark specification; and the publication of the benchmarking results.

The execution of a benchmark is a time consuming and complex process that needs the definition of clear and precise rules for performing benchmark tests and validate the benchmarking results. In order to support the definition of such process in LDBC, we analyze and discuss the process models presented in the organizations

initially mentioned (i.e. TPC, SPEC and W3C), and provide comments and recommendations. Particularly, we present guidelines concerning performance metrics, benchmark execution rules, and benchmarking results evaluation (i.e. auditing).

Document Information

IST Project Number	FP7 – 317548	Acronym	LDBC
Full Title	LDBC		
Project URL	http://www.ldbc.eu/		
Document URL	`http://www.ldbc.eu:8090/display/PROJECT/Deliverables`		
EU Project Officer	Carola Carstens		

Deliverable	Number	D1.1.2	Title	Benchmark principles and methods
Work Package	Number	WP1	Title	Common Benchmark Methodology

Date of Delivery	Contractual	M12	Actual	M12
Status	version 1.0		final ☒	
Nature	Report (R) ☒ Prototype (P) ☐ Demonstrator (D) ☐ Other (O) ☐			
Dissemination Level	Public (PU) ☒ Restricted to group (RE) ☐ Restricted to programme (PP) ☐ Consortium (CO) ☐			

Authors (Partner)	Renzo Angles (VUA)			
Responsible Author	Name	Renzo Angles	E-mail	r.anglesrojas@vu.nl
	Partner	VUA	Phone	+31619091465

Abstract (for dissemination)	The objective of this deliverable is to present an initial set of principles and methods for defining, performing, auditing and publishing RDF and graph database benchmarks in LDBC. We start the deliverable with a review of the process models defined by influential organizations dedicated to benchmarking and standardization (i.e. TPC, SPEC, and W3C). In our study, we put special interest in two main activities: the development of benchmarks, and the execution of benchmarks. Based on this study, we present a methodology for benchmark development which is guided by the identification and modeling of technological challenges called "choke points". Additionally we provide guidelines concerning performance metrics, benchmark execution rules, and benchmarking results evaluation (i.e. auditing).
Keywords	benchmarking, benchmark development process, choke points

Version Log			
Issue Date	Rev. No.	Author	Change
23/09/2013	0.1	Renzo Angles	First draft
30/09/2013	1.0	Renzo Angles	Final version

Table of Contents

List of Tables

1 Introduction

Benchmarking is a systematic and continuous process of creation, measurement, comparison and improvement. The process starts with the creation of a benchmark, which usually results in a collection of software tools and a specification with clear rules for executing the benchmark. Once the benchmark has been created, it can be used to measure the performance of the target systems. The results of executing the benchmark can be used, by vendors and consumers (systems users), to compare the performance of the evaluated systems. Finally, the comparison results cause improvements: vendors improve their technology in order to be competitive, and current benchmark design can be improved to cover new necessities and application domains.

In the previous deliverable D1.1.1, we presented the state of the art of benchmarks for core database functionalities, graph databases, RDF databases, and RDF data integration approaches. Additionally, we discussed benchmarking principles focusing mainly on the purpose and scope of benchmarks. The objective of the current deliverable is to present an initial set of principles and methods for defining, performing, auditing and publishing RDF and graph database benchmarks in LDBC.

In Section 2, we present a review of the process models define by influential organizations related to benchmarking, including the Transaction Processing Performance Council (TPC) and the Systems Performance Evaluation Cooperative (SPEC). In our study, we put special interest in two main processes inside such organizations: the benchmark development process, and the benchmark execution process. Additionally, we review the process defined by the World Wide Web Consortium (W3C) for the creation of Web standards. It is particularly interesting to understand the process model behind the creation of standards in an open and community oriented organization like the W3C.

Although several benchmarks for databases have been proposed, there is not a standard recipe behind their development. In Section 3, we present a methodology for developing database benchmarks in LDBC. We delve into methodology as the glue that binds all the elements that participate in the process of developing a benchmark and the guidelines that support such process.

The execution of a benchmark is a time consuming and complex process that needs the definition of clear and precise rules for performing and validating results. In order to support the definition of such process in LDBC, we analyze the process models presented in the organizations initially mentioned (i.e. TPC, SPEC and W3C), and provide guidelines (Section 4) concerning performance metrics, benchmark execution rules, and benchmarking results evaluation (i.e. auditing).

2 Process Models for Benchmarking

In this section we review the process models implemented in two well-know organizations related to benchmarking, these are TPC and SPEC. We focus on organization objectives, design principles, the process of benchmark development, and the process of benchmark execution. Additionally, we describe the process for Web standards creation applied by the W3C. The objective of this section is to provide background information that can be used in the definition of the benchmarking process in LDBC.

2.1 Transaction Processing Performance Council (TPC)

The Transaction Processing Performance Council (TPC) [6] is a nonprofit corporation created to provide the database community with benchmark standards, which measure performance in a standardized, objective and verifiable manner. The TPC's membership includes database software companies (e.g. Oracle), but also hardware companies in the database server business (e.g. Intel). The TPC's goal is to create, manage and maintain a set of fair and comprehensive benchmarks that enable end-users and vendors to objectively evaluate system performance under well-defined, consistent and comparable workloads [36].

The development of a TPC benchmark is a highly structured and lengthy process. The resulting specification is a dense document with a detailed description of the benchmark elements and rigorously define how the tests should be run, how system price should be measured, and how the results should be reported. Unique to the TPC is the requirement that all published benchmarks be audited by an independent third party, which has been certified by the organization. This requirement ensures that published results adhere to all of the specific benchmark requirements, and that results are accurate so any comparison across vendors or systems is, in fact, an "apples to apples" comparison.

The information presented in this section is based on the TPC Bylaws [12] and TPC policies [13] documents.

2.1.1 Roles and bodies

The following roles and bodies take part in the processes of developing and executing TPC benchmarks.

Members

The TPC considers two classes of members: full members and associate members. *Full Members* can participate in all the activities of the TPC, including the development of benchmark standards and voting on strategic decisions. *Associate members* are non-profit organizations (e.g. educational institutions) and businesses who do not create, market or sell computer products or services. Associate members are not admitted to vote on final approval of any and all proposed TPC Benchmark Standards. As an option, associate members may also join the TPC as members. Participation in technical subcommittees is voluntary and at the discretion of each member.

TPC may have *Professional Affiliates*, which are individuals designated by the TPC as engaged in business activity that complements or helps fulfill the TPC's mission. Affiliates cannot be full members or associate members, therefore, cannot make motions or vote on motions.

Board of Directors

The Corporation has one director (primary representative) for each member. The number of directors is only limited by the number of members. A Director of the Corporation may be any person, including a person who is a member, or an employee of a member, of the Corporation. The directors participate in the general meetings of the council.

Steering Committee (SC)

The Steering Committee consists of five representatives from the members. At least three members must be present to conduct business. All motions are passed by simple majority voting.

Technical Advisory Board (TAB)

The Technical Advisory Board consists of seven representatives from the Members. This board is responsible for providing analysis, definition and recommended resolution to requests for interpretations and benchmark standard compliance questions. At least four members must be present to conduct business. All motions are passed by simple majority voting.

Public Relations Committee (PRC)

The Public Relations Committee consists of five representatives from the members. This body is responsible for promoting the TPC, its charter, and its activities in the public arena (e.g. encouraging use of TPC benchmarks).

Benchmark Development Subcommittee

A development subcommittee is the body in charge of developing a benchmark. Throughout the benchmark development and approval process, the subcommittee owns: the benchmark specification, the TPC-Provided code, and the auditor exam for the TPC benchmark standard.

Benchmark Maintenance Subcommittees

A maintenance subcommittee is the body in charge of developing and recommending changes to an approved TPC Benchmark Standard. This body handle questions and requirements from the benchmark users (i.e. test sponsors). A maintenance subcommittee may make recommendations to the Council to change a benchmark specification, and will have the responsibility for creating a new version of the benchmark.

Auditors

An Auditor is an individual certified by the TPC to verify that the results of executing a benchmark meet the requirements of the appropriate TPC Benchmark Standard. The TPC has the responsibility to ensure that an adequate number of auditors is available to provide coverage in a timely manner. At the same time, the TPC has the authority to restrict the number of auditors to ensure high quality.

The following defines the certification process for audit candidates: application; exam and interview; apprenticeship; certification; and maintaining certification. Auditors and consultants are automatically granted Affiliate status when they are certified or hired, respectively.

2.1.2 The process of benchmark development

All TPC benchmark must satisfy the following standard requirements:

- define primary metrics selected to represent the workload being measured. The Primary Metrics must include both performance and price/performance metrics.
- include auditing requirements.
- include Executive Summary and Full Disclosure Report (FDR) requirements.

Design principles

The development of the TPC-C benchmark is based on six desirable attributes [27]:

- Relevant: The benchmark is meaningful within the target domain.
- Understandable: The benchmark is easy to understand and use.
- Good metrics: The metrics defined by the benchmark are linear, orthogonal and monotonic.
- Scalable: The benchmark is applicable to a broad spectrum of hardware and software configurations.
- Coverage: The benchmark workload does not oversimplify the typical environment.
- Acceptance: The benchmark is recognized as relevant by the majority of vendors and users.

Benchmark development process

The process of developing a TPC benchmark is highly structured so that valid assessments can be made across systems and vendors for any given benchmark. Before the release of any new benchmark standard, the TPC creates a lengthy and detailed definition of the new benchmark.

The benchmark development process in TPC consists of the following steps:

1. Benchmark submittal: One member elaborates and submits an initial benchmark proposal, named draft standard specification. The Council forwards the proposal to the steering committee for review. The steering committee analyzes the advantages and disadvantages of the proposal and proposes a course of action. The Council votes formally to accept or reject the proposal.
2. Creation of a development subcommittee: In case of accepting a proposal, the Council creates a benchmark development subcommittee. This subcommittee is responsible for the development of the benchmark and the corresponding specification.
3. Status and direction: During the development of a benchmark, the development subcommittee must provide status updates and working drafts to the Council, who provides direction and feedback.
4. Authorizing public release of draft specification: The development subcommittee is able to request the release of a draft specification. The objective is that members can review the draft, execute the benchmark, and provide feedback.
5. Accepting a standard for review: If the development subcommittee considers that a draft specification represents a high quality specification, it is submitted for approval to the Council.
6. Formal review: The Council and the members review the proposal and provide feedback. The specification can be updated with resolution of comments.
7. Approval for Mail Ballot: The Council must coordinate the mail ballot voting process.
8. Mail Ballot Approval: The approval of the specification is given by simple majority, i.e. greater than 50% of members present.
9. Creation of a Maintenance Subcommittee.

The final product of the process is a *TPC benchmark specification*, which is a written document that describes the elements of the benchmark (data and workload), and requirements for implementation, execution, auditing and reporting. A Specification may require the use of TPC-Provided Code. Such code may be source, executable binaries, or a combination of both. Note that source code may be provided without matching executable binaries and executable binaries may be provided without matching source code, as appropriate to each specification.

Benchmarking metrics

TPC benchmarks include performance, price/performance, and energy/performance metrics, which help customers identify systems that deliver the highest level of performance, using the least amount of energy. The TPC benchmarks (TPC-C, TPC-DS, TPC-E, TPC-H, TPC-VMS, TPC-Energy) use specific metrics depending of their application-domain.

- TPC-C measures performance in transactions per minute (tpm). The primary metrics are the transaction rate (tpmC), the associated price per transaction ($/tpmC), and the availability date of the priced configuration.
- The TPC-E metric is given in transactions per second (tps).
- TPC-H defined the Composite Query-per-Hour Performance Metric (QphH@Size), which reflects multiple aspects of the capability of the system to process queries (i.e. selected database size, query processing power, and query throughput). The TPC-H Price/Performance metric is expressed as $/QphH@Size.
- The TPC-Energy benchmark defines an energy metric to measure the energy consumption of system components associated with typical business information technology environments (i.e. servers, disk systems, and any other item that consume power).

The TPC-Pricing [5] specification provides consistent methodologies for reporting the purchase price of the benchmarked system, the licensing of software used in the benchmark, contracts for maintenance, and general availability. The TPC-Energy specification [4] defines a methodology and requirements for including and reporting energy consumption. The TPC- Pricing specification and TPC-Energy specifications are consistent across all current standards and are expected to be consistent across future specifications as well.

2.1.3 The process of benchmark execution

The TPC Policies [13] document provides a clear description of the rigorous process for results publishing and auditors policies.

A *Test Sponsor* is a company (not necessarily a member) that officially executes a TPC benchmark and submits a result. In order to evaluate a TPC benchmark over an specific database product, the test sponsor is required for accomplishing the following sequence of tasks:

- Implementing the TPC benchmark on their platform;
- Tuning the implementation of the benchmark;
- Auditing the implementation;
- Executing the benchmark on their configuration;
- Auditing the benchmark results;
- Filing the results report defined by TPC.

Auditing

The purpose of the audit is to verify compliance with TPC rules and the TPC benchmark specification. The audit encompasses more than just the benchmark test and includes a review of items that can affect the compliance of the benchmark.

All TPC benchmarking results must be audited by an independent third party, which has been certified by the TPC. The test sponsor must select an auditor from the list of auditors published by the TPC. The audit rate is determined by negotiation between the sponsor and the auditor, and must be paid by the sponsor.

The auditor determines the level of audit required and decides whether the audit or a portion of the audit requires his/her on-site presence at the test site. There are two levels of auditing:

- *Full Audit.* It makes no assumption of prior audits and requires full direct access to personnel and benchmark environment. This may require an on-site presence.

- *Updated Audit*. An updated audit leverages previous audits to a significant degree. This review is targeted at those components of the benchmark environment that have changed since the last implementation review. It requires a highly automated test environment. To audit the components which have changed, the auditor may require full direct access to personnel and benchmark environment.

The first step of the audit process is the verification of the benchmark implementation. During this step, called "pre-audit", all the components of the implementation are reviewed to verify that they satisfy the requirements defined by the TPC benchmark specification (it also includes correction of deficiencies and tuning of the implementation). All the elements (e.g. scripts) that will be used in execution of the benchmark must be approved by the auditor. After the pre-audit the benchmark can be executed and the benchmarking results are captured into predefined audit files. The auditing of the benchmarking results consists in the verification of the audit files. This step is called "remote audit" because it does not require physical presence of the auditor. When necessary, the auditor can remotely connecting to the test bed for additional hands-on verifications. The process is finished with the delivery of the fully audited results.

Hence, the audit process allows to:

- verify the compliance of all components of the implementation (e.g., software programs, hardware configurations, purchase and maintenance pricing, etc.).

- obtain a reasonable confidence level that the methodology used to implement the benchmark related tests produces documented results that demonstrate compliance.

- verify the compliance of each benchmark execution by examining the results produced during that execution.

- verify the compliance of the result based on applicable Technical Advisory Board and General Council rulings. This may require additions to the audit process to address issues not previously covered.

The audit does not guarantee compliance. In addition, there is a formal review process and a mechanism for determining compliance or non-compliance. The audit minimizes the probability that a benchmarking result will be found non-compliant in the review process.

Results reporting

Any TPC *Benchmarking Result* must be documented by two artifacts:

- *Executive summary.* A short document (two to four pages) that describes the configuration, primary metrics, performance data, and pricing details. The system configuration information must be correctly listed, including whether the configuration is of a particular type, e.g. c/s or cluster.

- *Full Disclosure Report (FDR)*. This must present all the information about the benchmarking result, and must include source code, configuration files and documentation for all the clauses defined by the benchmark standard.

Both documents must respect the requirements defined by the benchmark specification whereas the pricing specification for the pricing spreadsheet. A Test Sponsor must submit an electronic copy of the Executive Summary and the FDR to the Administrator the same calendar day the result is publicly disclosed.

Review process

The purpose of the review process is to enable all TPC members to examine the full disclosure report and understand the implementation of a benchmarking result. During the review period a result may also be changed and scrutinized until it gets the status of accepted (i.e. it has successfully completed a review process).

The review process begins when the result is posted to the Web Site. The result is subject to evaluation and challenge for a period of time defined as the *Review Period*. A result remains in *Submitted for Review* status during the review period. If no challenge is submitted to the Technical Advisory Board within the review period, the result is automatically given the status of Accepted. The full disclosure report is posted on the Web Site for review by all members, who may submit specific objections. TPC meetings can be required to discuss and solve the objections and challenges over the result. The result passes into *Accepted Status* when the following conditions are met: (i) All challenges accepted by Technical Advisory Board have been closed; (ii) The Council has not voted that the Result is "non-compliant'. (iii) No challenges are pending. In such case, the test sponsor is allowed to promote the result in the press. If the Council votes that a result is in "non-compliance" with the benchmark standard, the administrator immediately must withdraw that result from the TPC results list, and the test sponsor must stop publishing the result. The TPC requires that all Members and test sponsors follow the *Fair Use Policy* [13], which states how benchmarking results may be fairly used in public information.

2.2 Systems Performance Evaluation Cooperative (SPEC)

The Systems Performance Evaluation Cooperative (SPEC) is a nonprofit corporation of major computer vendors formed to establish, maintain and endorse a standardized set of relevant benchmarks that can be applied to the newest generation of high-performance computers [3].

The development of SPEC benchmark is based on the contributions from its members, including industry participants, independent testing labs, universities and consultants. SPEC provides several benchmark suites which are carefully designed to simulate real-world environments, packaged with source code and tools, and extensively tested for portability before release. Moreover, each benchmark suite establishes guidelines for the execution of the benchmark programs, including clear definition of exact system configurations.

2.2.1 Roles and bodies

The following roles and bodies take part in the processes of developing and executing SPEC benchmarks [44].

Members

Membership in SPEC is open to any interested hardware or software company or entity that is willing to commit to SPEC's standards. Membership is group-based (see below), therefore each member is responsible for paying the full dues and other responsibilities defined for each group. Membership gives/allows: full access to existing benchmarks (this included company-wide licenses for current benchmark products); full participation in the development of new benchmarks; free result publication through SPEC; to participate in general member voting.

SPEC also maintains an "Associate" category that enables academic and non-profit organizations to share in the SPEC process, but not to participate in general member voting. Additionally, the role of "Supporting Contributors" is given by invitation to a commercial organization, to an academic institution, or to an individual, to participate in the development of a single benchmark.

The Board of Directors

Each member is able to have a representative in the Board of Directors. This body manages the affairs of the corporation, exercise its powers, and control its property. Decisions of the board require agreement of the majority.

Groups

The groups are the bodies in charge of developing SPEC benchmarks. All groups operate under the bylaws and guidelines of SPEC. However, they run independently and is responsible for defining its ow rules for member rights and privileges; development of its own products; and meeting and voting rules.

Currently, there are four active groups in SPEC:

- The Open Systems Group (OSG): This group focuses on benchmarks for desktop systems, high-end workstations and servers running open systems environments.
- The High-Performance Group (HPG): Develops benchmarks to represent high-performance computing applications for standardized, cross-platform performance evaluation including symmetric multiprocessor systems, workstation clusters, distributed memory parallel systems, and traditional vector and vector parallel supercomputers.
- The Graphics and Workstation Performance Group (GWPG) : Develops consistent and repeatable graphics and workstation performance benchmarks and reporting procedures.
- SPEC Research Group (RG) : It was created to promote innovative research on benchmarking methodologies and tools facilitating the development of benchmark suites and performance analysis frameworks for established and newly emerging technologies.

Steering committee

The final decisions in a group are usually made by a "steering committee", which is elected by the membership of the group. The acceptance of a benchmark standard needs the ratification of all the members in the group.

2.2.2 The process of benchmark development

The development of a benchmark in SPEC is in charge of a working group created inside any of the current SPEC groups. The working group is responsible for creating the benchmark specification and providing a standardized suite of source code based upon existing applications. The use of already accepted and ported source code greatly reduces the problem of making apples-to-oranges comparisons.

Design principles

We identify the following benchmark design principles considered in SPEC:

- Application-oriented: SPEC benchmarks are designed to resemble end-user applications and aims to test "real-life" situations (as opposed to being synthetic benchmarks).
- Portability: SPEC benchmarks are written in a platform neutral programming language (usually C or Java).
- Repeatable and reliable: SPEC benchmarks must be designed to ensure that the experiments are repeatable and reliable.
- Consistency and fairness: Each SPEC benchmark specification must define clear rules for executing and reporting results.

The process

As mentioned before, each SPEC group must defines its own rules and guidelines for its operation, including its process for benchmark development. However, the policies and procedures defined for each group are based on ones defined by the Open Systems Group (OSG) [22], which was the original SPEC committee. Therefore, the process described in this section is based on the development process defined for the OSG.

SPEC maintains useful checklists for benchmark developers The process to approve the release of new benchmarks consists of several steps:

- *Conception*: A SPEC group sponsors the formation of a "working group", which must research the technologies of interest to determine the requirements for developing a new benchmark or to update older benchmark suites. All interested members and all non-profit associates are allowed to participate in the working group. A supporting contributor may also be invited.
- *Proposal*: The working group must prepare a report with a clear description of the new benchmark. This report must be delivered to the corresponding steering committee. The steering committee evaluates the report and decide if a subcommittee should be chartered to take up the task of producing the benchmark.
- *Beta version construction*: The subcommittee must prepare a beta version of the benchmark (including code and documentation). The beta version can be recommended for review to obtain feedback from the general membership.
- *Beta version review*: When the beta version of the benchmark is ready, the subcommittee must send out a final call for comments to the general membership. A member has four weeks to review and comment on the benchmark (after the call). The member should send their comments back to the subcommittee and the steering committee.
- *Subcommittee voting*: At the conclusion of the final call for comments period, the subcommittee will take a final vote to recommend that the group approve the benchmark for release. The subcommittee report to the group must include a report relative to the current release checklist.
- *SPEC Group voting*: After the group has received the subcommittee's report and recommendations, the group will review them, and must consider any comments from the general member review. The group will vote on whether to approve the benchmark for release or send it back to the subcommittee for further work.
- *Release*: With the approval of the Board of Directors, the benchmark approved within the OSG becomes an authorized SPEC Benchmark Suite release.

All SPEC benchmarks must provide the corresponding documentation [22], which must:

- clearly describe what the benchmark is intended to measure;
- provide introductory information;
- state hardware and software pre-requisites;
- describe how to install the benchmark product;
- describe how to use it;
- describe what is needed in order to have a complete disclosure for review by SPEC.

The documentation is required to be sufficiently complete so that a user with the appropriate technical knowledge can install and run the benchmark without direct assistance from developers of the benchmark.

Clear run and reporting rules must be defined for each SPEC benchmark. The rules should strive to ensure that results are meaningful, comparable, and reproducible, with full disclosure of relevant conditions. Subcommittees are encouraged to bring rules issues to the surface early, so that they can be discussed as fully and as impartially as possible. SPEC provides a template to assist with run rule development. Benchmark products may use automated mechanisms (for example, shell scripts, programs, and scripting languages) to control how benchmarks are run, to record system conditions, or to generate reports.

Benchmarking metrics

Measuring the performance of computer systems is a complex measure of the systems ability to perform a variety of operations like executing integer, floating-point, and I/O operations. SPEC benchmarks consider specific types of metrics, such as response time or throughput for a defined set of operations. For example, the SPECspeed metrics (e.g. SPECint2006) are used for comparing the ability of a computer to complete single tasks, and the SPECrate metrics (e.g., SPECint_rate2006) measure the throughput or rate of a machine carrying out a number of tasks. The available metrics and units for each benchmark (e.g. SPECjvm2008 Peak ops/m) are defined in the SPEC's Fair Use Policy [2] and within each benchmark run rules.

Two set of metrics can be measured with the benchmark suite, peak and base metrics. *Peak metrics* may be produced by building each benchmark in the suite with a set of optimizations individually selected for that benchmark, and running them with environment settings individually selected for that benchmark. The optimizations selected must adhere to the set of general benchmark optimization rules. *Base metrics* must be produced by building all the benchmarks in the suite with a common set of optimizations and running them with environment settings common to all the benchmarks in the suite [39].

2.2.3 The process of benchmark execution

SPEC establishes guidelines for the execution of the benchmark programs, including clear definition of exact system configurations, and reporting policies. Consistency and fairness are guiding principles for SPEC. Hence, any organization or individual who makes public use of SPEC benchmark results must follow the rules and guidelines defined by the SPEC's Fair Use Policy [2].

A test sponsor must pay for the use license of a SPEC benchmark. The costs vary from test to test with a typical range from several hundred to several thousand dollars. This pay-for-license model might seem to be in violation of the GPL as the benchmarks include software such as GCC that is licensed by the GPL. However, the GPL does not require software to be distributed for free, only that recipients be allowed to redistribute any GPLed software that they receive; the license agreement for SPEC specifically exempts items that are under "licenses that require free distribution", and the files themselves are placed in a separate part of the overall software package.

Results reporting

The SPEC suite provides execution rules, rules for reporting results and other guidelines that give performance results uniformity, clarity, credibility, and reproducibility.

In general, a SPEC result must satisfy the following requirements:

- Proper use of the SPEC benchmark tools as provided. To generate rule-compliant results, an approved tool-set must be used.
- Availability of an appropriate full disclosure report (FDR). A result must contain enough information to allow another user to reproduce the result.
- Availability of the Hardware and Software used.
- Support for all of the appropriate protocols and standards.

Furthermore, SPEC expects that any public use of results shall be for systems and configurations that are appropriate for public consumption and comparison. Thus, it is also expected that:

- Hardware and software used to run a SPEC benchmark must provide a suitable environment for the application area(s) targeted by the benchmark.
- Optimizations utilized must improve performance for a larger class of workloads than just the ones defined by this benchmark suite.

- The tested system and configuration is generally available, documented, supported, and encouraged by the providing vendor(s).

Review process

The results published for a benchmark must satisfy a peer review by the members of the group where the benchmark is developed. Each subcommittee is responsible for establishing a regular process for *peer review* of results, to help improve consistency in the understanding, application, and interpretation of the run rules. The result review process defined by the Open Systems Group is presented below as reference point.

At the beginning of the review cycle, a primary reviewer is assigned by the chair of the group to review the results. A primary reviewer is encouraged to offer his/her preliminary comments on the results within one week. At the end of the review cycle, the results are accepted or rejected by a majority vote of the committee. In some cases, the disposition of results may imply the need of corrections. Once a result is accepted, the full disclosure report is posted on the SPEC web site, hence the information contained in the result disclosure is considered public information. All results, including a comprehensive description of the hardware used in the tests are freely available.

2.3 World Wide Web Consortium (W3C)

The World Wide Web Consortium (W3C) is an international community created to develop Web standards. The W3C mission is to lead the World Wide Web to its full potential by developing protocols and guidelines that ensure the long-term growth of the Web. W3C's vision for the Web involves participation, sharing knowledge, and thereby building trust on a global scale. The most important work done by the W3C is the development of Web specifications (called "Recommendations") for several building blocks of the Web.

2.3.1 Roles and bodies

The following roles and bodies take part in the processes of developing W3C standards.

W3C Director

The W3C staff is led by Tim Berners-Lee, its current Director. He has an important role in the proposal and acceptance of Web standards.

Members

W3C functions is a member organization that includes a variety of software vendors, content providers, corporate users, telecommunications companies, academic institutions, research laboratories, standards bodies, and governments. The W3C members can be part of several committees and groups, participate in the Member Submission process, and access member-only information.

Advisory Committee

The Advisory Committee reviews formal proposals from the W3C Director (e.g. activity proposals, proposed recommendations, and proposed process documents). It is composed of one representative from each member organization.

Working Groups

The main goal of a Working Group is to produce deliverables (e.g., Recommendation Track technical reports, software, test suites, and reviews of the deliverables of other groups). There are three types of individual participants in a working group: member representatives, invited experts, and team representatives.

2.3.2 Principles

The development of W3C standards is based on two groups of principles: open standard principles, and design principles.

Open standards principles

On 29 August 2012 five leading global organizations agreed five principles in support of The Modern Paradigm for Standards:

- Cooperation: Respectful cooperation between standards organizations, whereby each respects the autonomy, integrity, processes, and intellectual property rules of the others.
- Adherence to Principles: five fundamental principles of standards development are considered: due process, broad consensus, transparency, balance openness.
- Collective Empowerment: Commitment by affirming standards organizations and their participants to collective empowerment.
- Availability: Standards specifications are made accessible to all for implementation and deployment.
- Voluntary Adoption Standards are voluntarily adopted and success is determined by the market.

Design principles

The following design principles guide W3C's work.

- Web for All: One of W3C's primary goals is to make these benefits available to all people. The social value of the Web is that it enables human communication, commerce, and opportunities to share knowledge.
- Web on Everything: The number of different kinds of devices that can access the Web has grown immensely.

2.3.3 Development process of W3C standards

A W3C Recommendation (the W3C equivalent of a Web standard) is a specification or set of guidelines that, after extensive consensus-building, has received the endorsement of W3C Members and the Director. The development and acceptation of a W3C recommendation is based on the consensus of the Membership, Team, and public.

Each W3C Recommendation is developed by a working group consisting of members and invited experts. The working group obtains its input from companies and other organizations. A W3C standard goes across the following maturity levels[1]: Working Draft (WD), Candidate Recommendation (CR), Proposed Recommendation (PR), and W3C Recommendation (REC).

The process of developing a W3C recommendation is summarized as follows:

- A group of W3C members submits a suggestion for a Web standard to the consortium.
- When a submission is acknowledged by the W3C, the Director announces the development of a proposal for a new Activity or Working Group charter.
- The working group elaborates a Working Draft which is published for review by the community, including W3C Members, the public, and other technical organizations.

[1] `http://www.w3.org/2005/10/Process-20051014/tr.html`

- When the W3C believes that a Working Draft has been widely reviewed and satisfies the Working Group's technical requirements, then it is considered a Candidate Recommendation. A Candidate Recommendation can be published to gather implementation experience (i.e. testing from members and software vendors.).
- When the Director considers that the working group has fulfilled the general requirements for advancement and shown that each feature of the Candidate Recommendations has been implemented, the maturity report is named a Proposed Recommendation, and is set to the W3C Advisory Committee for final review.
- The maturity technical report is considered a W3C Recommendation when, after extensive consensus-building, there is significant support from the Director, the Advisory Committee, the Team, W3C Working Groups, and the public. The decision to advance a document to Recommendation is a W3C decision (see below).

Advisory committee review

The objective of the Advisory Committee Review is to decide the approval or rejection of a Candidate Recommendation.

The Advisory Committee review period begins with a Call for Review from the Team to the Advisory Committee. Each W3C member may send one review, and may also share their reviews with other members on the Advisory Committee. After the review period, the Director must announce to the Advisory Committee the level of support for the proposal:

- Consensus: A substantial number of individuals in the set support the decision and nobody in the set registers a formal objection. Unanimity is the particular case of consensus where all individuals in the set support the decision and no individual in the set abstains.
- Dissent: At least one individual in the set registers a formal objection.

A *W3C decision* is one where the Director has exercised the role of assessing consensus after an Advisory Committee review. Where unanimity is not possible, a group should strive to make consensus decisions where there is significant support and few abstentions. The W3C does not require a particular percentage of eligible participants to agree to a motion in order for a decision to be made. A group charter may define formal voting procedures (e.g., quorum or threshold requirements) for making decisions about substantive issues. When a group is unable to reach consensus (i.e., there is at least one formal objection), then the Director may record a decision so that the group may make progress.

A W3C decision concerning a Proposed Recommendation is generally one of the following:

- The proposal is approved, possibly with minor changes integrated.
- The proposal is approved, possibly with substantive changes integrated.
- The proposal is returned for additional work, with a request to the initiator to formally address certain issues.
- The proposal is rejected.

Advisory Committee representatives may appeal certain decisions, though appeals are only expected to occur in extraordinary circumstances.

3 Methodology for benchmark development in LDBC

A benchmark is normally composed of four main elements: (1) the data schema, which defines the structure of the data used by the benchmark; (2) the workload, which defines the set of operations that the system under benchmarking has to perform during the benchmark execution; (3) performance metrics, which are used to measure (quantitatively) the performance of the systems; and (4) execution rules, which are defined to assure that the results from different executions of the benchmark are valid and comparable.

Literature until now has described the technical work required when designing a good database system benchmark in relatively vague terms. LDBC intends to formalize some of the best practices and raise the state-of-the-art in this area, in its guidelines for benchmark development.

3.1 Roles and bodies

The following roles and bodies take part in the processes of developing and executing LDBC benchmarks.

The board of LDBC members

This is formed by one representative (director) per each member organization of LDBC with voice and one single vote per organization in their assembly. Meetings of the directors make all policy decisions, though certain decisions, among which the adoption of new benchmarks, is handled through a written vote and requires an absolute majority of all members.

Technical User Community (TUC)

The TUC is an open organization that brings together users of graph and RDF technologies, researchers, industry participants, and delegates of the LDBC members in physical events (TUC meetings) to discuss about possible benchmark use cases and scenarios and to assess the quality of the benchmark proposals and the adequacy to their needs. LDBC organizes multiple TUC meetings per year, providing logistics and travel assistance to external invitees who require this. The TUC also has a web portal in which members can share information[1].

Task Forces

The Task Force is an internal LDBC structure that carries the development of a benchmark from beginning to end. It is formed by experts from member organizations of LDBC, and it works on the proposal, creation of the use case and scenario based on the TUC discussions, choke points suggested by technical experts and the implementation of the different parts of the benchmark (e.g. data and workload generation).

3.2 Design principles

In the literature on benchmarking we can find several definitions about what a good benchmark means, all of them based on the fulfillment of specific attributes or properties (see Appendix A). Among these attributes we can mention:

- In general, a good benchmark [47]: is written in a high-level language, making it portable across different machines; is representative of some programming style or application; can be measured easily; has wide distribution.
- According to Jim Gray, a domain-specific benchmark must meet four important criteria [21]: relevance, portability, simplicity and scalability.

[1] LDBC TUC web portal: `http://www.ldbc.eu:8090/display/TUC/Technical+User+Community`

- The development of the TPC-C benchmark is based on six desirable attributes [27]: relevant, understandable, good metrics, scalable, coverage and acceptance.
- Huppler [23] proposed five key aspects that all good benchmark must have: relevant, repeatable, fair, verifiable and economical.
- In the context of Big data benchmarking, a successful benchmark would be simple to implement and execute, cost effective, timely and verifiable [1].
- In the context of knowledge representation, we found the following principles that should guide the evaluation process for an ontology matching system [25]: systematic procedure, continuity, quality, equity and dissemination.

Most authors coincide that a good benchmark should not satisfy all the possible attributes, because their adoption is determined by several factors like the application domain and the objectives of the benchmark. The most important thing is to select a set of attributes that reflects the purpose of the benchmark. Moreover, it is important to understand the compromises made to enable one strength over another than it is to satisfy every possible criterion [23].

LDBC will create benchmarks that will follow a set of important principles based on the literature [21, 23] and its own research.

- **Relevance and simplicity**. The benchmarks will be based on the actual requirements of the TUC, and they will be created on the base of realistic use case scenarios, making sure that their results will be understandable by users outside the TUC. At the same time they will be designed to be simple in concept and to allow the end user to understand the workload and results easily.
- **Push on technologies.** The benchmarks will be designed to push the available technologies by focusing on the technical challenges that limit their today's usefulness. For this, LDBC introduces the concept of choke point as the specific technological difficulties that will be forced by a benchmark.
- **Repeatability and portability**. The benchmarks will provide a set of expected outcomes for each of the queries and reporting guidelines that will allow for the repetition of a specific execution. At the same time, the benchmarks will be portable along different software data management platforms and hardware systems.
- **Fairness**. The benchmarks will be fair making sure that no vendor benefits from specific parts of the workload. This will be assured by the board of LDBC members, and by a fair selection of delegates in the Task Force carrying on a specific benchmark.
- **Verifiable**. The reporting system designed by LDBC, auditing procedures and supporting materials provided for each benchmark, will allow for verifiable results in each reported execution.
- **Scalability**. The benchmarks created will provide data generators that will allow for scaling the datasets up to platforms that actually execute on very large databases. Also, the workloads will be thought in such a way that their complexity is not an impediment when scaling to such large scenarios.
- **Economically sustainable**. The benchmarks designed will make a special emphasis on the sustainability of each implementation. In this sense, they will make sure that they can be implemented in a short amount of time and that the reporting for them is clear and requires a short time for its provision.

3.3 Choke point based design.

On the surface, a benchmark models a particular scenario, and this should be believable, in the sense that users of the benchmark must be able to understand the scenario and believe that this use-case matches a larger class

of use cases appearing in practice. On a deeper – technical – level, however, a benchmark exposes technology to a workload. Here, a benchmark is valuable if its workload stresses important technical functionality of actual systems. This stress on elements of particular technical functionality we call "choke points". To understand benchmarks on this technical level, intimate knowledge of actual system architectures is needed. The LDBC consortium was set-up to gain access to those architects of the initial LDBC industry members, as well as to the architects of database systems RDF-3X, HyPer, MonetDB and Vectorwise.

In a recent paper [10], LDBC authors analyzed the relational TPC-H benchmark in terms of 28 different choke points; providing both a good illustration of the choke point concept, and an interesting to-do list for those optimizing a system for TPC-H. Specific examples among those 28 are choke points like exploiting functional dependencies in group-by, foreign-key joins with a low match ratio (to be exploited by e.g. bloom filters), and discovering correlation among key attributes in a clustered index (e.g. using zone maps). The complete analysis is presented in Appendix B.

Choke points can be an important design element during benchmark definition. The technical experts in a task force identify choke points relevant for a scenario, and document these explicitly. Subsequently, as the benchmark workload evolves during the process of its definition, a close watch is kept on which queries in the workload test which choke point to which extent, aiming for complete coverage using a limited amount of queries. Choke points thus can ensure that existent techniques are present in a system, but can also be used to reward future systems that improve performance on still open technical challenges.

3.4 Benchmark development process

In this section we define the process of benchmark development, that is the phases of the process, the steps that need to be performed in each phase and how the phases relate and influence each other. The proposed process consider the phases of conceptualization, analysis, design, implementation, testing and distribution of the benchmark.

3.4.1 Conceptualization

The objective of this phase is to formulate a general idea of the benchmark by means of its high-level features including application domain, purpose and use-case scenarios.

The conceptualization of the benchmark use case scenarios will be decided thanks to the collaboration of the TUC, that will put their needs through in regular meetings organized by the consortium. TUC members will be informed about the progress of the design and implementation of the benchmarks, and asked for feedback and ideas regarding their needs and opinions of the benchmarks under development.

Selection of the application domain

It means the selection of an area of interest for commercial and/or scientific communities related to data management. It must include a description of characteristics considered as relevant for the development of the benchmark. Examples of application domain are social network and semantic publishing.

Definition of the purpose of the benchmark

It implies to define the reasons for the development of the benchmark. The most general reason for the creation of a benchmark in LDBC will to stimulate the advances in the current technologies. Additionally, specific reasons must be specified according to the scope of the benchmark.

Selection of use case scenarios

This step implies the selection of distinguishing scenarios, where the systems of the application domain have great expectations to have a growing market. The selected scenarios must be clear, understandable and relevant for the users and communities (commercial and scientific). For example, scientific analysis has emerged as

an important scenario, applicable in several application domains, where chemists analyze chained reactions or biologists protein interactions.

3.4.2 Analysis

This phase is oriented to determine the requirements of the benchmark based on the analysis of the application domain, workload characterization, selection and definition of choke points, and specification of the data schema.

Analysis of the application domain

It implies the definition of a consistent understanding of the target application domain. We must analyze and describe all the elements that participate of the application domain, including actors and processes. It is essential to describe the current approaches and technologies in the application domain, including principles, techniques, deficiencies and current problems.

Workload characterization

A workload is a collection of operations or tasks that the system under benchmarking has to perform during the benchmark execution. Workload characterization is one of the most general, central and difficult problem in computer system performance evaluation. The performance cannot be seriously and meaningfully measured unless the workload is carefully selected [17].

The characterization process started with the selection of an application domain scenario during the conception of the benchmark. The roles, use cases and interactions of the scenario should be analyzed in order to identify classes of workload. The identified classes are measure in order to determine their impact in the application domain. The result should be the definition of real-life workloads.

Definition of choke points

The behavior of a real workload can be very complex and difficult to reproduce. The goal is to define a workload, called the *driver workload*, which captures the static and dynamic behavior of the real workload. The implementation of the benchmark is based on the characteristics of the driver workload.

Our approach to define the driver workload is based on the identification and analysis of choke points. A *choke-point* is an unsolved technical challenge which limit the usefulness of today's technology. For example, query execution and optimization in large graphs is strongly affected by value/structure correlations. Additionally, graph and RDF queries typically consist of many more joins than in relational query optimization, making this much more challenging.

The identification of choke points comes from the analysis and discovery of limitations in existing systems. The identified choke points must be analyzed to demonstrate their feasibility. For this reason, preliminary experiments should be performed to make sure that the selected choke points could be overcome. Finally, the choke points should be classified regarding the expected impact in the application domain. For example some limitations might be considered acceptable if no dataset is expected to trigger these limitations, while other limitations might be severe even for common datasets.

Definition of the data schema

The data schema (also called conceptual schema or conceptual data model) defines the structure of the data in terms of entities and their relationships. In this context is recommendable the use of abstract models for data representation, for example the entity-relationship model or its UML equivalent representation. The data schema must be defined following the requirements imposed by the driver workload and the choke points. In particular, special properties and relationships in the data must be identified, analyzed and described, for example data consistency, data correlations and statistical distributions for the data.

3.4.3 Design and implementation

The objective of this phase is the design and implementation of solutions according to the benchmark requirements. It implies the elaboration of a design document where the solutions and methods for data and workload generation are described.

In this phase diverse software tools and applications are designed and implemented, particularly the data generator and the workload generator. It also includes the definition of the metrics for measuring performance, as well as rules for benchmark execution and reporting of results.

Data generation

The objective of this step is the definition of a method for data generation that can be used to construct the datasets used by the benchmark. The datasets must be generated according to the requirements imposed by the benchmark data schema, the driver workload and the choke points. In particular, special properties and relationships in the data must be specified and implemented (e.g. data consistency, data correlations and statistical distributions).

The most common approach to this step is the design and implementation of a data generator that enables the creation of synthetic data but including properties of real-life data. A synthetic data generator must be simple, easy-to- use, configurable and scalable.

Workload design

The goal of this step is to build a workload generator that provides a collection of operations organized in a systematic way. Each operation forces a specific feature or weak point defined by a choke point.

Besides the operations in the workload, it is also relevant to define the methods for test data selection. *Test data* refers to the data specifically selected, from the generated dataset, to be used as substitution parameters for the operations of the workload. The selection of test data is strongly determined by the choke points, and their quality is essential to the execution of the workload.

Considering that the data generator, the workload generator and the methods for test data generation are mutually depended, the definition of these elements follows an iterative process of analysis, design and implementation.

Execution rules

The rules for benchmark execution consider the following aspects: the total dataset size must be disclosed; any modifications to the benchmark configuration file must be disclosed; the hardware used to execute the benchmark must be fully disclosed, including (CPU type, chipset, physical memory, hard disks (and their configuration), network adapters, O/S, etc.) as well as total cost at full list price.

3.4.4 Testing and distribution

This phase is oriented to prepare the benchmark to be successfully released. This phase consider the activities of verification (testing), validation and distribution.

The *verification* of the benchmark ensures that the implementation fulfill the functional requirements of the benchmark. It implies that the features of the benchmark must be tested to ensure that errors are detected and solved. In the *validation* step, the benchmark will be executed on a number of systems and the results will be analyzed to check that the benchmark meets its principles and purpose. This experimentation thus provides insight in how far the benchmark indeed tests the choke points that were targeted.

After the benchmark is validated, it can be distributed. It includes all the operation to package the test drivers, data-sets and/or synthetic data generators, documentation (including execution, auditing and reporting rules). The benchmark package must contains all the information necessary to run the benchmark. When a benchmark is eventually launched, companies will be able to run it on their binomial computer/software.

4 The Process of Benchmark execution in LDBC (Guidelines)

The task of measuring performance is a time consuming and complex process. For this reason, a benchmark specification must provide clear and useful rules and guidelines for benchmark execution and results reporting.

In this section we analyze the process models described in Section 2. and discuss their rules and methods for benchmark execution. The objective is to provide some advise and suggestions for the definition of the benchmarking process in LDBC.

Benchmarks are used to compare the performance of software and hardware technologies. For this reason, the performance results of a benchmark must be characterized by uniformity, clarity, credibility, and reproducibility. In this sense, fundamental components of any benchmark specification are: performance metrics, execution rules, and results evaluation rules.

4.1 Performance metrics

Computer systems performance is difficult to quantify because there are several factors affecting the operation of the systems, and these factors vary depending of the application areas. A performance metric is the basis for assessing the performance of the systems. Several performance metrics have been defined and used in database benchmarking. The most basic metrics are *response time* (i.e. the time to execute an operation) and *throughput* (i.e. number of operations executed per unit of time). Scaling the average response time or the throughput with a cost measure results in *cost/performance metrics* (e.g. cost per transaction per second, K$/TPS). Additionally, speedup and scale-up are widely accepted metrics for evaluating the performance of parallel database systems [16].

The performance metric used by a database benchmark determine its applicability, suitability and credibility. Therefore, performance metrics must be selected carefully. As a starting point, we can consider the following proposal of desirable characteristics for performance metrics [29]:

- Linearity: It means that the value of the performance metric should be linearly proportional to the actual system performance. That is, if the valued of the metric changes by a ratio, the actual performance of the system should change by the same radio.
- Reliability: A performance metric is reliable if a system A always outperform system B when the corresponding values of the metric for both systems indicate that system A should outperform system B (i.e. larger value better performance)
- Repeatability: A performance metric is repeatable if the same value of the metric is measured each time the experiment is performed. It implies that a good metric is deterministic.
- Easy to measure: If a metric is not easy to measure, it is unlikely that anyone will actually use it.

The issue of finding a "universal metric" has been largely discussed in the literature [18, 43, 19, 26, 30, 11, 14]. The more general recommendation is to put special attention to the type of data is under review and be aware that we can not measure everything. We must consciously decide what pieces we would approach and what pieces we wouldn't (though some of them seem interesting to measure) [45]. Moreover, we must select metrics that are able to scale and at the same time conserve the desired properties defined above.

LDBC benchmarks should include performance metrics as well as cost-based metrics (price/performance). Additionally, LDBC can consider new approaches according to the requirements in current technologies and application domains. For example, a metric for *robustness* can be used to measure the ability of a database to perform well under a variety of conditions, including adverse runtime conditions such as unexpected data skew or resource contention [48]. Such metric can based on the following characteristics: graceful degradation, to measure how special conditions impact the performance; consistency, to measure how performance vary across conditions; and optimality, to measure how performance vary between implementations. Robustness comparisons can be very useful to motivate technology improvements.

4.2 Execution rules

In this section we shall briefly examine criteria to consider when running benchmarks. A benchmark is not defined only by its data and workload. The conditions under which measurements are to be made are an indispensable element of a benchmark definition. Failure to clearly define the conditions opens different ways of abusing the metric.

Generally, the execution rules need to take a stand on the following points:

Warm vs cold working set

Throughput can vary by a factor of 100 between running from memory and running from secondary storage. Running from secondary storage cached by the OS can still be several times slower than running from data resident in the DBMS process. There is no single universal way of enforcing any particular cache behavior since these things are highly system dependent.

There are two main approaches to managing this:

1. Running for long enough tends to establish a steady state with respect to working set. The measurement window can be started after the system is in steady state and needs to be long enough to guarantee that a representative quantity of work is done and that this throughput could in principle be sustained indefinitely. This is typical of transaction benchmarks, e.g. TPC-C, TPC-E.

2. Specifying a sequence of operations that must be carried out without or with highly restricted intermediate actions. For example, bulk load the database, optionally gather statistics, ensure persistence, run the workload. This is typical of analytics benchmarks like TPC-H and TPC-DS.

Both approaches result in running from memory in during the measurement window if memory is sufficient.

Number of iterations in executing the workload

The intent of a performance metric is to indicate a level of performance a system can sustain in principle indefinitely. If the data grows during the run, which is typically the case at least in OLTP benchmarks, it follows that the system must be sized to accommodate a realistic increase in data size, e.g. 180 days worth of data accumulated at the reported throughput in the case of TPC-C.

Enforcing ACID during the run

A special case of the indefinitely sustainable throughput is managing durability of new data. Normally durability is provided by transaction logging. However a transaction log that grows indefinitely consumes indefinite space and is anyhow unusable since its roll forward would take indefinitely long during database recovery.

So, if significant data is written, the execution rules specify that a certain number of log checkpoints need to take place during a run. A log checkpoint is a system-dependent operation which makes the changes persisted in a transaction log persistent in the persistent database structures, so that the transaction log can be deleted or archived after the checkpoint. In some cases durability can be provided by redundancy but present (TPC) benchmark practice does not consider this.

In analytical benchmarks up to several percent of the data change during an execution. The data maintenance is generally tied to how many concurrent users are simulated, so as to provide a steady read/write ratio., TPC-H and TPC-DS do expect durability, typically provided by transaction logging but do not expect a checkpoint, i.e. it is sufficient that the changes were durable only through logging at the end of the benchmark run.

Coverage of dataset

An OLTP-style benchmark should be required to run long enough in order to have touched essentially all data that would be touched if the system ran indefinitely. Therefore for lookup-style benchmarks the number of transactions to execute must be tied to the dataset size.

Coverage of exception states

OLTP transactions encounter diverse exceptions during production. For example, a TPC-C new order transaction processes a new order where the items concerned are most often supplied from one warehouse. In 1% or order lines, the warehouse will however be different from the warehouse from which the other lines are supplied. Since the warehouse is a natural partitioning key, this variation is significant, as it will cause 10% of new orders to be distributed transactions in a scale out system.

Further, transactions will not always commit. In TPC-C, 1% of new orders are required to be aborted because of bad input data. In OLTP workloads a certain amount of data contention is expected, e.g. there will be waiting for locked rows or pages or there will be update conflicts aborting a transaction in optimistic concurrency control situations.

Benchmark design must take resource contention into account in the workload definition and the run rules need to make sure the intended exceptions occur approximately as frequently as expected. Analytical benchmarks do not have exceptions in the same sense. Data maintenance can be scheduled to run without lock contention and all operations are expected to conclude without a retry.

4.3 Auditing

The most influential benchmarking organizations (like TPC and SPEC) enforce the development of objective benchmark specifications that provide:

1. *vigorous execution rules*, that ensure that a benchmark is satisfactorily implemented and executed, over a system under-test correctly configured; and

2. *strict reporting rules* based on the "full disclosure" clause, that ensure transparency, credibility and repeatability of results (i.e. all the details about data, workload, metrics, optimizations, scripts, hardware/software configuration must be reported and published in order to make the test reproducible and the results comparable).

These strictness and disclosure requirements can be tightened even more by adding external **auditing** and internal **reviews**. The practice of auditing is inseparable from the question of implementation and execution rules this auditing is intended to enforce.

The TPC has introduced the practice of only publishing audited benchmark results where a third party certified by the benchmark authority (TPC) verifies that a benchmark result is produced according to all the rules set forth in the benchmark definition. The auditing process was instantiated to make sure that database vendors do not make misleading or false performance claims, as well as to maintain the credibility of the TPC benchmarks.

The auditor's responsibilities are divided between static aspects of a benchmark implementation, e.g. code, query text, schema, SUT configuration and run time aspects, e.g. sequence of operations, consistent configuration of the SUT during different parts of a benchmark run, absence of queries that might inform the SUT on the workload to follow etc. Due to the above, the auditor must be well conversant with best practices of database administration, tuning, query optimization and generally database application development.

Benchmarks fall in the broad categories of analytics and OLTP. In analytics benchmarks the implementation is usually relatively fixed, as in TPC-H or TPC-DS. In OLTP benchmarks the test implementor has broader choice of means, e.g. use of stored procedures. Certainly in the graph database space benchmark implementations will be quite heterogeneous due to lack of standard query language. Hence the auditor will have to inspect more code than in a typical TPC benchmark implementation.

Auditing a benchmark may involve tasks that are not directly associated with the benchmark run itself. For example, the auditor may have to verify product functionality, such as transaction ACID compliance and failure recovery capabilities of the SUT. For this purpose, additional test scripts may be used that primarily focus on ensuring e.g. the durability of transaction and the consistency of concurrent reads modifications. These tests in case of TPC also include the step of the auditor physically disconnecting the electricity from the server while

running an update workload to verify that the system recovers consistently without data loss, and/or the on-line removal of redundant system parts such as a hard drive that is part of a RAID, while running a consistency-check test workload.

Benchmark metrics typically include a price/performance indicator. It is the auditor's responsibility to verify that the equipment actually used is correctly represented in the report and that the prices quoted for the software and hardware are realistic, i.e. what any normal customer would pay when buying said items in single quantity and without special discounts.

There is a tradeoff between the thoroughness of auditing and the cost and difficulty of producing a benchmark result. The TPC has been for example criticized for having make benchmarks expensive to run (in terms of time and resources), and hard to publish. This criticism is not however exclusively aimed at auditing. In different situations, different degrees of auditing thoroughness will make sense. For example, auditing gives credibility to the benchmarking organization.

The possibilities range among the following:

1. The auditor performs the entire run on the premises of the test sponsor.
2. The test sponsor performs the run and the auditor checks compliance according to the rules in the benchmark definition. This is done on site, since there are verification steps that can require physical access to the SUT. For example, powering down by pulling the plug or removing storage from a running system.
3. The auditor follows the test sponsor's run over remote login.
4. The auditor only reviews the code and claims of the test sponsor without witnessing the run itself.

TPC benchmarks are run in scenario 2. For LDBC, to reduce the cost of publishing results, there is a general understanding of going by model 3, i.e. remote login.

REFERENCES

[1] Big Data Benchmarking Community. http://clds.sdsc.edu/bdbc.

[2] SPEC Fair Use Rule. http://www.spec.org/fairuse.htmlspe.

[3] Standard Performance Evaluation Corporation (SPEC). http://www.spec.org.

[4] TPC-Energy Specification. http://www.tpc.org/tpc_energy/.

[5] TPC-Pricing Specification. http://www.tpc.org/pricing/.

[6] Transaction processing performance council (tpc). http://www.tpc.org/.

[7] Daniel J. Abadi. Query execution in column-oriented database systems. MIT PhD Dissertation, 2008. PhD Thesis.

[8] Daniel J. Abadi, Samuel Madden, and Nabil Hachem. Column-stores vs. row-stores: how different are they really? In *SIGMOD Conference*, pages 967–980, 2008.

[9] Cagri Balkesen, Jens Teubner, Gustavo Alonso, and M. Tamer Özsu. Main-memory hash joins on multi-core cpus: Tuning to the underlying hardware. In *ICDE*, 2013.

[10] Peter Boncz, Thomas Neumann, and Orri Erling. TPC-H Analyzed: HIdden Messages and Lessons Learned from an Influential Benchmark. In *TPC Technology Conference on Performance Evaluation and Benchmarking (TPCTC)*, 2013.

[11] Daniel Citron, Adham Hurani, and Alaa Gnadrey. The harmonic or geometric mean: does it really matter? *SIGARCH Comput. Archit. News*, 34(4):18–25, 2006.

[12] Transaction Processing Performance Council. TPC Bylaws (Version 2.7). http://www.tpc.org/information/about/documentation/spec/bylaws_v2.7.pdf, March 12 2013.

[13] Transaction Processing Performance Council. TPC Policies (Version 5.26). http://www.tpc.org/information/about/documentation/spec/TPC_Policies_v5.26.pdf, June 1 2013.

[14] Alain Crolotte. Issues in benchmark metric selection. In *TPC Technology Conference on Performance Evaluation and Benchmarking (TPCTC)*, 2009.

[15] Jérôme Darmont. Database Benchmarks. In *Database Technologies: Concepts, Methodologies, Tools, and Applications*, pages 1226–1233. 2009.

[16] David DeWitt and Jim Gray. Parallel database systems: the future of high performance database systems. *Communications of the ACM*, 35(6):85–98, 1992.

[17] D. Ferrari. Workload characterization and selection in computer performance measurement. *IEEE Computer*, 5(4):18–24, 1972.

[18] Philip J. Fleming and John J. Wallace. How not to lie with statistics: the correct way to summarize benchmark results. *Communications of the ACM*, 29(3):218–221, 1986.

[19] Ran Giladi and Niv Ahituv. SPEC as a Performance Evaluation Measure. *Computer*, 28(8):33–42, 1995.

[20] Goetz Graefe. Query evaluation techniques for large databases. *ACM Comput. Surv.*, 25(2):73–170, 1993.

[21] Jim Gray, editor. *The Benchmark Handbook for Database and Transaction Systems (2nd Edition)*. Morgan Kaufmann, 1993.

[22] SPEC Open Systems Group. Policies and Procedures (Version 2.17). http://www.spec.org/osg/policy.html, August 23 2013.

[23] Karl Huppler. The Art of Building a Good Benchmark. In *TPC Technology Conference on Performance Evaluation and Benchmarking (TPCTC)*, LNCS 5895, pages 18–30, August 2009.

[24] Ihab F. Ilyas, Volker Markl, Peter J. Haas, Paul Brown, and Ashraf Aboulnaga. Cords: Automatic discovery of correlations and soft functional dependencies. In *SIGMOD Conference*, pages 647–658, 2004.

[25] J.Euzenat and P. Shvaiko, editors. *Ontology Matching*. Springer-Verlag, 2007.

[26] Lizy Kurian John. More on finding a single number to indicate overall performance of a benchmark suite. *SIGARCH Comput. Archit. News*, 32(1):3–8, 2004.

[27] Charles Levine. TPC-C: The OLTP Benchmark. In *SIGMOD International Conference on Management of Data - Industrial Session*, 1997.

[28] Quanzhong Li, Minglong Shao, Volker Markl, Kevin S. Beyer, Latha S. Colby, and Guy M. Lohman. Adaptively reordering joins during query execution. In *ICDE*, pages 26–35, 2007.

[29] David J. Lilja. *Measuring Computer Performance - A Practitioner's Guide*. Cambridge University Press, 2005.

[30] John R. Mashey. War of the benchmark means: time for a truce. *SIGARCH Comput. Archit. News*, 32(4):1–14, 2004.

[31] Guido Moerkotte. Small materialized aggregates: A light weight index structure for data warehousing. In *VLDB*, pages 476–487, 1998.

[32] Guido Moerkotte and Thomas Neumann. Dynamic programming strikes back. In *SIGMOD Conference*, pages 539–552, 2008.

[33] Guido Moerkotte and Thomas Neumann. Accelerating queries with group-by and join by groupjoin. *PVLDB*, 4(11):843–851, 2011.

[34] Fabian Nagel, Peter Boncz, and Stratis D. Viglas. Recycling in pipelined query evaluation. In *ICDE*, 2013.

[35] Raghunath Othayoth Nambiar and Meikel Poess. The making of TPC-DS. In *VLDB*, pages 1049–1058, 2006.

[36] Raghunath Othayoth Nambiar, Meikel Poess, Andrew Masland, H. Reza Taheri, Matthew Emmerton, Forrest Carman, and Michael Majdalany. TPC Benchmark Roadmap 2012. In *TPC Technology Conference on Performance Evaluation and Benchmarking (TPCTC)*, number 7755 in LNCS, pages 1–20. Springer, August 27 2012.

[37] Thomas Neumann and Guido Moerkotte. A framework for reasoning about share equivalence and its integration into a plan generator. In *BTW*, pages 7–26, 2009.

[38] Thomas Neumann and Gerhard Weikum. Scalable join processing on very large RDF graphs. In *Proceedings of the 35th SIGMOD international conference on Management of data*, pages 627–640. ACM, 2009.

[39] David Padua, editor. *Encyclopedia of Parallel Computing*. Springer, 2011.

[40] J. Rao, B. Lindsay, G. Lohman, H. Pirahesh, and D. Simmen. Using EELs: A practical approach to outerjoin and antijoin reordering. In *ICDE*, pages 595–606, 2001.

[41] Praveen Seshadri, Hamid Pirahesh, and T. Y. Cliff Leung. Complex query decorrelation. In *ICDE*, pages 450–458, 1996.

[42] David E. Simmen, Eugene J. Shekita, and Timothy Malkemus. Fundamental techniques for order optimization. In H. V. Jagadish and Inderpal Singh Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996*, pages 57–67. ACM Press, 1996.

[43] J. E. Smith. Characterizing computer performance with a single number. *Communications of the ACM*, 31(10):1202–1206, 1988.

[44] Standard Performance Evaluation Corporation (SPEC). Corporate Bylaws. http://www.spec.org/spec/docs/spec_bylaws.pdf, September 2010.

[45] Jack Stephens and Francois Raab. TPC-D Detailed Description (Interview). http://www.tpc.org/tpcd/detail.asp.

[46] Marco Vieira and Henrique Madeira. From performance to dependability benchmarking: A mandatory path. In *TPC Technology Conference on Performance Evaluation and Benchmarking (TPCTC)*, volume 5895 of *LNCS*, pages 67–83. Springer, August 2009.

[47] Reinhold P. Weicker. An overview of common benchmarks. *Computer*, 23(12):65–75, December 1990.

[48] Janet L. Wiener, Harumi Kuno, and Goetz Graefe. Benchmarking query execution robustness. In *TPC Technology Conference on Performance Evaluation and Benchmarking (TPCTC)*, 2009.

[49] Marcin Zukowski, Niels Nes, and Peter A. Boncz. DSM vs. NSM: Cpu performance tradeoffs in block-oriented query processing. In *DaMoN*, pages 47–54, 2008.

A Desirable attributes of a benchmark

In this section we compile the definitions, found in the literature, about desirable attributes for a benchmark.

Acceptance. Vendors and users embrace the benchmark [27].

Adaptability. A benchmark must be able to propose various database or workload configurations, to allow experiments to be performed in various conditions[15].

Continuity This principle dictates that the benchmark should evolve in order to incorporate the progress made in the field to include new challenges. For instance, in the case of query processing, the query workload should evolve to reflect the features that are added to the (preferably standard) query language used to express the queries [25].

Dissemination. The benchmark along with the results from different engines, should be publicly and freely available and disseminated in order to be used by the different stakeholders from both academia and industry. In this way the benchmark will be more easily adopted and become a standard [25].

Economical / Cost effective. So that the benefits of executing the benchmark justify its expense [1]. Economical does not mean "cheap", but rather "worth the investment" [23].

Fair / Equity. All systems and/or software being compared with the benchmark can participate equally [23]. The datasets and workloads to be used should not be biased towards a specific system, but should be driven only by the tasks to be solved [25].

Intelligible / Understandable / Comprehensible. A benchmark must be understandable, otherwise it will lack credibility [21, 27]. It dictates that the benchmark must specify how the results are represented so that they could be easily analyzed and understood by everyone [25].

Quality. Depending on the problem in hand, good quality may refer to different things related to the datasets and test cases (i.e., queries, workloads) of the benchmark [25].

Relevant. A benchmark is relevant when is meaningful within the target domain [27]. There are a number of characteristics that can make a benchmark relevant, or irrelevant[23]: Meaningful and understandable metric; Stresses software features in a way that is similar to customer applications; Exercises hardware systems in a way that is similar to customer applications; Longevity (leading edge but not bleeding edge); Broad applicability; Does not misrepresent itself; Has a target audience that wants the information.

Repeatable / Reproducible. There is confidence that the benchmark can be run a second time with the same result [23].

Representative / Coverage. A benchmark does not oversimplify the typical environment of the target domain [27]. A benchmark must be as representative as possible of a given domain but, as an abstraction of that domain, it will always be an imperfect representation of reality [46].

Simple. A successful benchmark would be simple to implement and execute [1].

Portable. It should be easy to implement the benchmark on many different systems and architectures [21].

Scalable. A benchmark must be applicable to a broad spectrum of hardware architecture (small and large computer systems) [27]. It should be possible to scale the benchmark up to larger systems, and to parallel computer systems as computer performance and architecture evolve [21].

Datasets (either real or synthetic) should scale in order to test different systems. For query processing, queries should return results of different sizes thereby testing the limitations of the query engine. Moreover, the query response times of all queries in the workload as data scales should evolve in the same way. In general, systems should not exhibit a behavior where for different data sizes, different queries of a workload dominate over others, hence having different benchmarks for different data sizes [25].

Timely. With benchmark versions keeping pace with rapid changes in the marketplace [1].

Verifiable. There is a high degree of confidence that the documented result represents the actual performance of the system under test [23]. The results of the benchmark can be validated via independent means [1].

B TPC-H Analyzed: A Post-Mortem Choke Point Analysis

This core text of this chapter was presented in the Fifth TPC Technology Conference on Performance Evaluation & Benchmarking (TPCTC 2013)

he LDBC project introduces the concept of "choke point" based benchmark development, where system architects deliberately inject well-chosen technical challenges in a realistic workload. LDBC authors argue that good benchmark contain good choke-points, and thus already existing high-quality benchmarks already contain choke-points, even if the benchmark design had been more intuitive and choke-point based design had not been an explicit ingredient of the benchmark design process. To illustrate this, we analyze the TPC-H benchmark, which has been the most important benchmark for analytical database workloads in the past two decades. From this, we find that TPC-H contains at least 28 choke-points, in six general areas of database functionality, which have visibly influenced the design of state-of-the-art analytical database technology.

B.1 Introduction

The TPC-D benchmark was developed almost 20 years ago, and even though its current existence as TPC-H could be considered superseded by TPC-DS, one can still learn from it. We focus on the technical level, summarizing the challenges posed by the TPC-H workload as we now understand them, which we call "choke points". We identify 28 different such choke points, grouped into six categories: Aggregation Performance, Join Performance, Data Access Locality, Expression Calculation, Correlated Subqueries and Parallel Execution. On the meta-level, we make the point that the rich set of choke-points found in TPC-H sets an example on how to design future DBMS benchmarks.

Good benchmark design starts with a use case that is recognizable and understandable, and where the data being stored as well as query and update workloads being posed, resemble those of a wider class of data management problems faced by IT practitioners (and more, see [23]). However, basing a benchmark solely on "real-life" data management scenarios, data-sets and query logs will not necessarily lead to an interesting benchmark, for instance because such real-world examples characterize what technology can do now, not what it could do in the future. Moreover, the value in a benchmark is not only in allowing data management practitioners to test different technologies and compare them quantitatively, but also in stimulating *technological advances*.

In the LDBC (Linked Data Benchmark Council) project, these authors are currently pursuing the design of new benchmarks that will stimulate technological advance in graph (and RDF) data management. For this purpose, LDBC follows a dual design track where on the one hand a Technical User Community (TUC) consisting of data management technology practitioners contribute data-sets and workloads, but on the other hand, technology experts both from industry and academic database research provide technical guidance on what we call "choke points", that should be embedded in these new benchmarks. Choke points are those technological challenges underlying a benchmark, whose resolution will significantly improve the performance of a product.

This analysis was written with a dual motivation: (i) to use the by now well-understood TPC-H benchmark to illustrate examples of what we understand "choke points" to be, and use TPC-H as an example of a benchmark that contains a rich set of these, and (ii) as an overview and reference for analytical data management practitioners to better understand the TPC-H workload itself; concentrating collected wisdom on this benchmark in a single place.

We do not dispute that TPC-H, which is almost 20 years old, could by some be regarded as superseded (e.g. by TPC-DS). The purpose of this analysis is *not* to criticize TPC-H or suggest improvements as has been done elsewhere [35], but rather to describe what TPC-H is. We would appreciate any future benchmark to be at least as rich in relevant technical challenges as TPC-D was in 1995.

B.2 TPC-H Choke Point Analysis

Table B.1 contains the summary of our choke point classification, which in the remainder of this section will be discussed point-by-point.

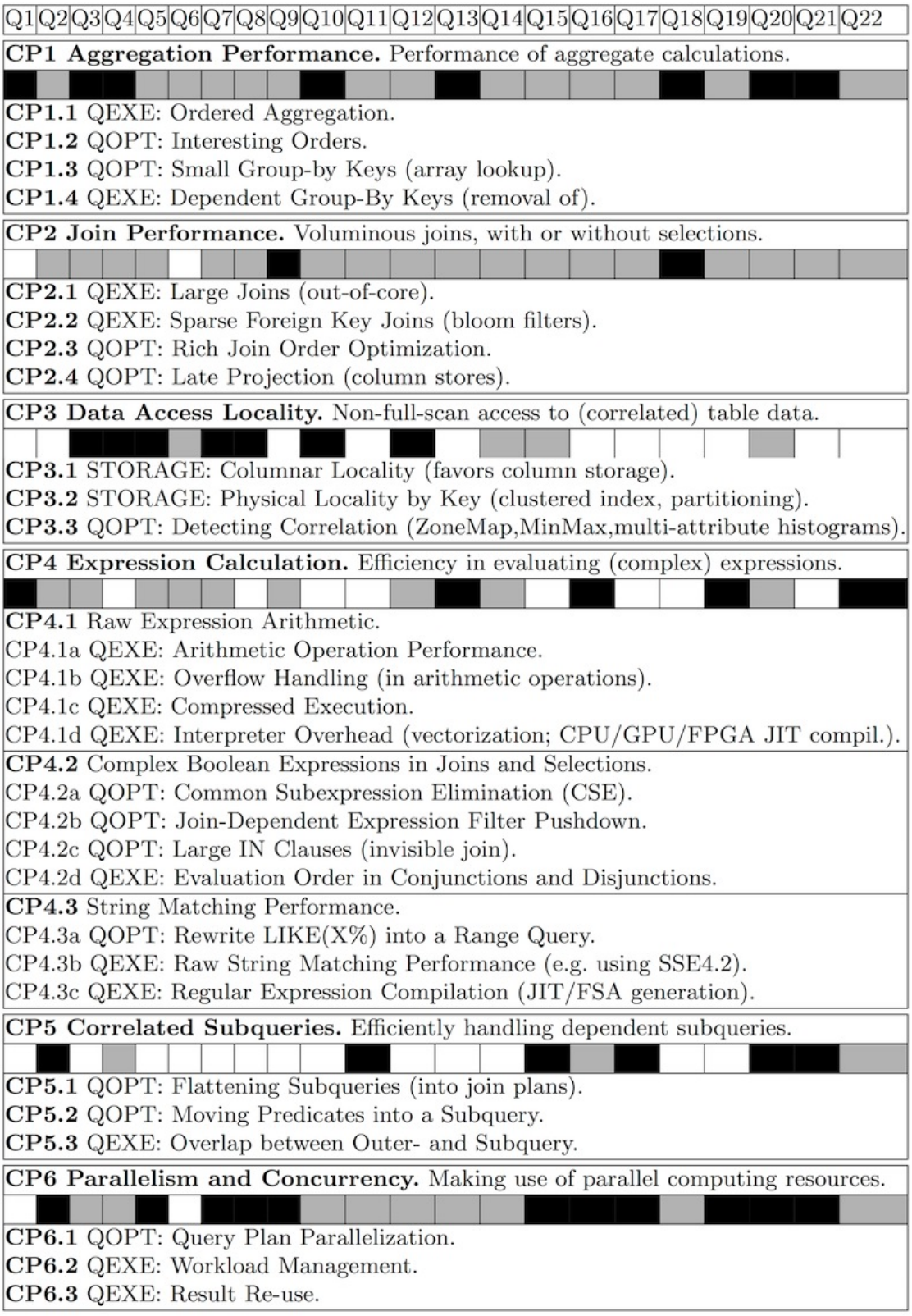

Table B.1: TPC-H Choke Point (CP) classification, and CP impact per query (white=light, gray=medium, black=strong).

B.2.1 CP1: Aggregation Performance

Aggregations occur in all TPC-H queries, hence performance of group-by and aggregation is quite important.

CP1.1: Ordered Aggregation

Aggregation implementations typically use a hash-table to store the group-by keys in. This is an efficient method, because hash-lookup (with a properly sized hash-table) has constant lookup cost. Hash-aggregation does run into performance deterioration when the amount of distinct group-by keys is large. When the hash-table will no longer fit the various CPU cache levels, cache and TLB misses will make the lookup more costly CPU-wise. With even more distinct keys, one may get to the situation that the hash-table cannot be kept in RAM anymore. Here a *spilling* hash aggregation would be needed, that first hash-partitions the tuple stream to different files based on the hash value, and then aggregates the individual files inside RAM one-at-a-time. Spilling hash aggregations are not obviously superior to other methods, such as those based on creating a B-tree or, more plausibly, those based on sorting (external memory sort). In case the group-by keys arrive in sorted order, or actually much more generally, if all equal group-by keys *appear consecutively* in the stream, one should employ ordered aggregation instead of hash aggregation.

These approaches can even be mixed, e.g., using repetitive grouped execution of hash-aggregation, or using hash-based early aggregation in a sort-based spilling approach. Therefore the key challenge is detecting which situation applies, which depends both on the available hardware and the query characteristics. Related to this, the query optimizer has to infer the correct intermediate result cardinalities, which is relatively simple for most TPC-H query constructs, but challenging for group-by expressions.

CP1.2: Interesting Orders

Apart from clustered indexes providing key order, other operators also preserve or even induce tuple orderings. Sort-based operators create new orderings, typically the probe-side of a hash join conserves its order, etc. For instance TPC-H Q3,4,18 join `ORDERS` and `LINEITEM`, followed by aggregation grouped-by on `o_orderkey`. If the tuple order of `ORDERS` is conserved by the join, ordered aggregation is applicable. This is not to say that it is always best to use the join order with `ORDERS` on the probe side and `LINEITEM` on the build side (in hash-join terms), but *if* this is chosen then the ordered aggregation benefit should be reaped. A similar opportunity arises in Q21 with a join between `SUPPLIER` and `LINEITEM`, and grouped-by on `s_suppkey`. These are an examples of *interesting order* handling where the query optimization space should take multiple orders into account [42] (i.e. choosing a particular join methods leads to lower aggregation cost, subsequently).

CP1.3: Small Group-By Keys

Q1 computes eight aggregates: a count, four sums and three averages. Group-by keys are `l_returnflag, l_linestatus`, with just four occurring value combinations. This points to a possibility to optimize a special case of group-by. Namely, if all group-by expressions can be represented as integers in a small range, one can use an array to keep the aggregate totals by position, rather then keeping them in a hash-table. This can be extended to multiple group-by keys if their concatenated integer representation is still "small". In case of Q1, the group-by attributes are single-characters strings (VARCHAR(1)) which can be stored as an integer e.g. holding the Unicode value.

CP1.4: Dependent Group-By Keys

Q10 has a group-by on `c_custkey` *and* the columns `c_comment`, `c_address`, `n_name`, `c_phone`, `c_acctbal`, `c_name`. The amount of data processed is large, since the query involves a one-year `ORDERS` and `LINEITEM` join towards `CUSTOMER`. Given that `c_custkey` is the primary key of `CUSTOMER`, the query optimizer can deduce that its value *functionally determines* the columns `c_comment`, `c_address`, `n_name`, `c_phone`, `c_acctbal`, `c_name`. As a result, the aggregation operator should have the ability to exclude certain group-by attributes from key matching: this can greatly reduce the CPU cost and (cache) memory footprint of such an operator.

This opportunity arises in many other queries that have an aggregation that includes a tuple identity (denoted #) in addition to other columns that are functionally determined by it:

Q3 #o → o_shippriority, o_orderdate

Q4 #o → o_orderpriority

Q10 #c → c_comment, c_address, n_name, c_phone, c_acctbal, c_name

Q13 #c → count(*)

Q18 #c,#l → l_quantity, o_totalprice, o_orderdate, c_name

Q20 #s → s_address, s_name

Q21 #s → s_name

Even though declaring keys is optional in the rules of TPC-H, functional dependency exploitation in aggregation is a clear argument why one would do so. An additional argument is execution optimization that can be performed when executing N:1 foreign key joins: knowing that exactly one value will be added to an intermediate result record, allows to lower CPU effort (breaking off hash-table search after the first hit) and to avoid intermediate data copying, which is needed if a join "blows up" an intermediate result in case of a 1:N join.

In this sense, it is noteworthy that the EXASOL TPC-H implementations do not declare (foreign) keys, but add a "foreign key check" query set to the load phase; it is understood that a side effect of this may be the detection of these (foreign) key constraints. This might avoid the only drawback of declaring constraints: namely the obligation to check these in the refresh queries.

B.2.2 CP2: Join Performance

CP2.1: Large Joins

Joins are the most costly relational operators, and there has been a lot of research and different algorithmic variants proposed. Generally speaking, the basic choice is between hash- and index-based join methods. It is no longer assumed that hash-based methods are always superior to index-based methods; the choice between the two depends on the system implementation of these methods, as well as on the physical database design: in general, index-based join methods are used in those situations where the data is stored in an index with a key of which the join key is a prefix. For the cost model, whether the index is clustered or unclustered makes a large difference in systems relying on I/O; but (as by now often is the case) if the TPC-H workload hot-set fits into the RAM, the unclustered penalty may be only moderate.

Q9 and Q18 are the queries with the largest joins *without* selection predicates between the largest tables ORDERS and LINEITEM. The heaviest case is Q9, which essentially joins it also with PARTSUPP, PART and SUPPLIER with only a 1 in 17 selection on PART. The join graph has the largest table LINEITEM joining with both ORDERS and PARTSUPP. It may be possible to get locality on the former join, using either clustered indexing or table partitioning; this will create a merge-join like pattern, or a partitioned join where only matching partitions need to be joined. However, using these methods, the latter join towards the still significantly large PARTSUPP table will not have locality. This lack of locality causes large resource consumption, thus Q9 can be seen as the query that tests for out-of-core join methods (e.g. spilling hash-joins). In TPC-H, by configuring the test machine with sufficient RAM, typically disk spilling can be avoided, avoiding its high performance penalty. In the case of parallel database systems, lack of join locality will cause unavoidable network communication, which quickly can become a performance bottleneck. Parallel database systems can only avoid such communication by replicating the PARTSUPP, PART and SUPPLIER tables on all nodes – a strategy which increases memory pressure and disk footprint, but which is not penalized by extra maintenance cost, since the TPC-H refresh queries do not modify these particular tables.

For specific queries, usage of special join types may be beneficial. For example, Q13 can be accelerated by the GroupJoin operator [33], which combines the outer join with the aggregation and thus avoids building the same hash table twice.

CP2.2: Sparse Foreign Key Joins

Joins occur in all TPC-H queries except Q1,6; and they are invariably over N:1 or 1:N foreign key relationships. In contrast to Q9 and Q18, the joins in all other queries typically involve selections; very frequently the :1 side of the join is restricted by predicates. This in turn means that tuples from the N: side, instead of finding exactly one join partner, often find no partner at all. In TPC-H it is typical that the resulting *join hit-ratios* are below 1 in 10, and often much lower. This makes it beneficial for systems to implement a *bloom filter* test inside the join [20]; since this will eliminate the great majority of the join lookups in a CPU-wise cheap way, at low RAM investments. For example, in case of VectorWise, bloom filters are created on-the-fly if a hash-join experiences a low hit ratio, and make the `PARTSUPP-PART` join in Q2 six times faster, accelerating Q2 two-fold overall.

Bloom filters created for a join should be tested as early as possible, potentially before the join, even moving it down into the probing scan. This way, the CPU work is reduced early, and column stores may further benefit from reduced decompression cost in the scan and potentially also less I/O, if full blocks are skipped [38]. Bloom filter pushdown is furthermore essential in MPP systems in case of such low hit-ratio joins. The communication protocol between the nodes should allow a join to be preceded by a bloom filter exchange; before sending probe keys over the network in a communicating join, each local node first checks the bloom filter to see if it can match at all. In such way, bloom filters allow to significantly bring down network bandwidth usage, helping scalability.

CP2.3: Rich Join Order Optimization

TPC-H has queries which join up to eight tables with widely varying cardinalities. The execution times of different join orders differ by orders of magnitude. Therefore, finding an efficient join order is important, and, in general, requires enumeration of all join orders, e.g., using dynamic programming. The enumeration is complicated by operators that are not freely reorderable like semi, anti, and outer joins. Because of this difficulty most join enumeration algorithms do not enumerate all possible plans, and therefore can miss the optimal join order. One algorithm that can properly handle semi-, anti-, and outer-joins was developed by IBM for DB2 [40]. Moerkotte and Neumann [32] presented a more general algorithm based on hypergraphs, which supports all relational operators and, using hyperedges, supports join predicates between more than two tables.

CP2.4: Late Projection

In column stores, queries where certain columns are only used late in the plan, can typically do better by omitting them from the original table scans, to fetch them later by row-id with a separate scan operator which is joined to the intermediate query result. Late projection does have a trade-off involving locality, since late in the plan the tuples may be in a different order, and scattered I/O in terms of tuples/second is much more expensive than sequential I/O. Late projection specifically makes sense in queries where the late use of these columns happens at a moment where the amount of tuples involves has been considerably reduced; for example after an aggregation with only few unique group-by keys, or a top-N operator. There are multiple queries in TPC-H that have such pattern, the most clear examples being Q5 and Q10.

A lightweight form of late projection can also be applied to foreign key joins, scanning for the probe side first only the join keys, and only in case there is a match, fetching the remaining columns (as mentioned in the bloom filter discussion). In case of sparse foreign key joins, this will lead to reduced column decompression CPU work, and potentially also less I/O – if full blocks can be skipped.

B.2.3 CP3: Data Access Locality

A popular data storage technique in data warehousing is the *materialized view*. Even though the TPC-H workload consists of multiple query runs, where the 22 TPC-H queries are instrumented with different parameters, it is possible to create very small materialized views that basically contain the parameterized answers to the queries. Oracle issued in 1998 the One Million Dollar Challenge, for anyone who could demonstrate that Microsoft SQLserver 7.0 was not 100 times slower than Oracle when running TPC-D; exploiting the fact that Oracle had introduced materialized views before SQLserver did. Since materialized views essentially turn the

decision support queries of TPC-D into pre-calculated result-lookups, the benchmark no longer tested ad-hoc query processing capabilities. This led to the split of TPC-D into TPC-R (R for Reporting, now retired, where materialized views were allowed), and TPC-H, where materialized views were outlawed. As such, even though materialized views are an important feature in data warehousing, TPC-H does not test their functionality.

CP3.1: Columnar Locality

The original TPC-D benchmark did not allow the use of vertical partitioning. However, in the past decade TPC-H has been allowing systems that uniformly vertically partition all tables ("column stores"). Columnar storage is popular as it accelerates many analytical workloads, without relying on a DBA to e.g. carefully choose materialized views or clustered indexes. As such, it is considered a more "robust" technique. The main advantage of columnar storage is that queries only need to access those columns that actually are used in a query. Since no TPC-H query is of the form `SELECT * FROM ..`, this benefit is present in all queries. Given that roughly half of the TPC-H data volume is in the columns `l_comment` and `o_comment` (in VectorWise), which are very infrequently accessed, one realizes the benefit is even larger than the average fraction of columns used by a query.

Not only do column-stores eliminate unneeded I/O, they also employ effective columnar compression, and are best combined with an efficient query compiler or execution engine. In fact, both the TPC-H top-scores for cluster and single-server hardware platforms in the years 2010-2013 have been in the hands of columnar products (EXASOL and VectorWise).

CP3.2: Physical Locality by Key

The TPC-H tables `ORDERS` and `LINEITEM` contain a few date columns, that are correlated by the data generator:

- `l_shipdate` = `o_orderdate` + random[1:121],
- `l_commitdate` = `o_orderdate` + random[30:90], and
- `l_receiptdate` = `l_shipdate` + random[1:30].

In Q3, there is a selection with lower bound (LO) on `l_shipdate` and a higher bound (HI) on `o_orderdate`. Given the above, one could say that `o_orderdate` is thereby restricted on the day range [LO-121:HI]. Similar bounds follow for *any* of the date columns in `LINEITEM`. The combination of a lower and higher bound from *different* tables in Q3 is an extreme case, but in Q4,5,8,10,12 there are range restrictions on one date column, that carry over to a date restriction to the other side of the `ORDERS`-`LINEITEM` join.

Clustered Indexes. It follows that storing the `ORDERS` relation in a clustered index on `o_orderdate` and `LINEITEM` on a clustered index on any of its date columns; in combination with e.g. unclustered indexes to enforce their primary keys, leads to joins that can have high data locality. Not only will a range restriction save I/O on both scans feeding into the join, but in a nested-loops index join the cursor will be moving in date order through both tables quasi-sequentially; even if the access is by the `orderkey` via an unclustered index lookup. Such an unclustered index could easily be RAM resident and thus fast to access.

In Q3,4,5,8,10,12 the date range selections take respectively 2,3,12,12,3,12 out of 72 months. Typically this 1 in 6 to 1 in 36 selection fraction on the `ORDERS` table is propagable to the large `LINEITEM` table, providing very significant benefits. In Q12 the direction is reverted: the range predicate is on `l_receiptdate` and can be propagated to `ORDERS` (similar actually happens in Q7, here through `l_shipdate`). Even though this locality automatically emerges during joins if `ORDERS` and `LINEITEM` both are stored in a clustered index with a date key, the best plan might not be found by the optimizer if it is not aware of the correlation. Microsoft SQLserver specifically offers the `DATE_CORRELATION_OPTIMIZATION` setting that tells the optimizer to keep correlated statistics.

Table Partitioning. Range-partitioning is often used in practice on a time dimension, in which case it provides support for so-called *data life-cycle management*. That is, a data warehouse may keep the X last months of data, which means that every month the oldest archived month must be removed from the dataset. Using range-partitioning, such can be efficiently achieved by range-partitioning the data per month, dropping the oldest partition. However, the refresh workload of TPC-H does not fit this pattern, since its deletes and inserts are not time-correlated. The benefit from table partitioning in TPC-H is hence *partition pruning*, which both can happen in handling selection queries (by not scanning those partitions that cannot contain results, given a selection predicate) and in joins between tables that are partitioned on the primary and foreign keys.

Data correlation could be exploited in partitioning as well, even respecting the TPC-H rule that no index creation directive (or any other DDL) would mention multiple tables. For example, for range-partitioned tables it is relatively easy to automatically maintain for all declared foreign key joins to another partitioned table a *pruning bitmap* for each partition, that tells with which partitions on the other side the join result is empty. Such a pruning bitmap would steer join partition pruning and could be cheaply maintained as a side effect of foreign-key constraint checking.

CP3.3: Detecting Correlation

While the TPC-H schema rewards creating these clustered indexes, in case of `LINEITEM` the question then is which of the three date columns to use as key. One could say that `l_shipdate` is used more often (in Q6,15,20) than `l_receiptdate` (just Q12), but in fact it should not matter which column is used, as range-propagation between correlated attributes of the same table is relatively easy. One way is through creation of multi-attribute histograms after detection of attribute correlation, such as suggested by the CORDS work in DB2 [24]. Another method is to use small materialized aggregates [31] or even simpler MinMax indexes (VectorWise) or zone-maps (Netezza), The latter data structures maintain the MIN and MAX value of each column, for a limited number of zones in the table. As these MIN/MAX are rough bounds only (i.e. the bounds are allowed to be wider than the real data), maintenance that only widens the ranges on need, can be done immediately by any query without transactional locking.

With MinMax indexes, range-predicates on any column can be translated into qualifying tuple position ranges. If an attribute value is correlated with tuple position, this reduces the area to scan roughly equally to predicate selectivity. For instance, even if the `LINEITEM` is clustered on `l_receiptdate`, this will still find tight tuple position ranges for predicates on `l_shipdate` (and vice versa).

B.2.4 CP4: Expression Calculation

TPC-H tests expression calculation performance, in three areas:

- **CP4.1: raw expression arithmetic**.
- **CP4.2: complex boolean expressions in joins and selections**.
- **CP4.3: string matching performance**.

We elaborate on different technical aspects of these in the following.

Q1 calculates a full price, and then computes various aggregates.[1] The large amount of tuples to go through in Q1, which selects 99% of `LINEITEM`, makes it worthwhile to optimize its many arithmetic calculations.

[1] **Some notes on Q1**. Compared to Q6, the only other non-join query, the amount of computation done in Q1 is larger, making it more likely to be CPU-bound than Q6. Also, Q1 trivially parallelizes: the aggregate result is very small, so the plan can be run on many cores (or machines) in parallel without need for synchronization or result communication of any significance. This makes Q1 the only query that allows to make back-of-the-envelope estimates of the computational power of a database engine even across systems and platforms and database sizes, since normalization to a single-core and scale is relatively straightforward.

CP4.1a: Arithmetic Operator Performance

According to the TPC-H rules, it is allowed to represent decimals as 64-bits doubles, yet this will lead to limited SIMD opportunities only (4-way in 256-bit AVX). Moreover, this approach to decimals is likely to be unacceptable for business users of database systems, because of the rounding errors that inevitably appear on "round" decimal numbers. Another alternative for decimal storage is to use variable-length numerical strings, allowing to store arbitrarily precise numbers; however in that case arithmetic will be very slow, and this would very clearly show in e.g. Q1.

A common and efficient implementation for decimals is to store integers containing the number without dot. The TPC-H spec states that the decimal type should support the range [-9,999,999,999.99: 9,999,999,999.99] with increments of 0.01. That way, the stored integer would be the decimal value times 100 and 42-bits of precision are required for TPC-H decimals, hence a 32-bits integer is too small but a 64-bits integer suffices. Decimal arithmetic can thus rely on integer arithmetic, which is machine-supported and even SIMD can be exploited.

It is not uncommon for database systems to keep statistics on the minimum and maximum values in each table column. The columns used in Q1 exhibit the following ranges: `l_extendedprice`[0.00:100000.00], `l_quantity`[1.00:50.00], `l_discount`[0.00:0.10] and `l_tax`[0.00:0.08]. This means that irrespective of how data is physically stored (columnar systems would typically compress the data), during query processing these columns could be represented in byte-aligned integers of 32, 16, 8 and 8 bits respectively. The expression `(1-l_discount)` using an 8-bits representation can thus be handled by SIMD subtraction, processing 32 tuples per 256-bits AVX instruction. However, the subsequent multiplication with `l_extendedprice` requires to convert the result of `(1-l_discount)` to 32-bits integers, still allowing 256-bits SIMD multiplication to process 8 tuples per instruction. This highlights that in order to exploit SIMD well, it pays to keep data represented as long as possible in as small as possible integers (stored column-wise). Aligning all values on the widest common denominator (the 32-bits `extendedprice`) would hurt the performance of four out of the six arithmetic operations in our example Q1; making them a factor 4 slower.

While SIMD instructions are most easily applied in normal projection calculations, it is also possible to use SIMD for updating aggregate totals. In aggregations with group-by keys, this can be done if there are multiple `COUNT` or `SUM` operations on data items of the same width, which then should not be stored column-wise but rather row-wise in adjacent fields [49].

CP4.1b: Overflow Handling

Arithmetic overflow handling is a seldom covered topic in database research literature, yet it is a SQL requirement. Overflow checking using if-then-else tests for each operation causes CPU overhead, because it is extra computation. Therefore there is an advantage to ensuring that overflow cannot happen, by combining knowledge of data ranges and the proper choice of data types. In such cases, explicit overflow check codes that would be more costly than the arithmetic itself can be omitted, and SIMD can be used. The 32-bits multiplication `l_extendedprice*(1-l_discount)`, i.e. [0.00:100000.00]*[0.00:0.90] results in the more precise value range [0.0000: 90000.0000] represented by cardinals up to 900 million; hence 32-bits integers still cannot overflow in this case. Thus, testing can be omitted for this expression.

CP4.1c: Compressed Execution

Compressed execution allows certain predicates to be evaluated without decompressing the data it operates on, saving CPU effort. The poster-child use case of compressed execution is aggregation on RLE compressed numerical data [7], however this is only possible in aggregation queries without group-by. This only occurs in TPC-H Q6, but does not apply there either given that the involved `l_extendedprice` column is unlikely to be RLE compressed. As such, the only opportunities for compressed execution in TPC-H are in column vs. constant comparisons that appear in selection clauses; here the largest benefits are achieved by executing a VARCHAR comparison on dictionary-compressed data, such that it becomes an integer comparison.

CP4.1d: Interpreter Overhead

The large amount of expression calculation in Q1 penalizes slow interpretative (tuple-at-a-time) query engines. Various solutions to interpretation have been developed, such as using FPGA hardware (KickFire), GPU hardware (ParStream), vectorized execution (VectorWise) and Just-In-Time (JIT) compilation (HyPer, ParAccel); typically beating tuple-at-a-time interpreters by orders of magnitude in Q1.

CP4.2a: Common Subexpression Elimination

A basic technique helpful in multiple TPC-H queries is common subexpression elimination (CSE). In Q1, this reduces the six arithmetic operations to be calculated to just four. CSE should also recognize that two of the average aggregates can be derived afterwards by dividing a `SUM` by the `COUNT`, both also computed in Q1.

CP4.2b: Join-Dependent Expression Filter Pushdown. In Q7 and Q19 there are complex join conditions which depend on both sides of the join. In Q7, which is a join query that unites customer-nations (`cn`) via orders, lineitems, and suppliers to supplier-nations (`sn`), and on top of this it selects:

```
(sn.n_name = '[NATION1]' AND cn.n_name = '[NATION2]') OR
(sn.n_name = '[NATION2]' AND cn.n_name = '[NATION1]').
```

Hence TPC-H rewards optimizers that can analyze complex join conditions which cannot be pushed below the join, but still derive *filters* from such join conditions. For instance, if the plan would start by joining `CUSTOMER` to `NATION`, it could immediate filter the latter scan with the condition:

```
(cn.n_name = '[NATION1]' OR cn.n_name = '[NATION2]')
```

This will reduce data volume by a factor 12.5. A similar technique can be used on the disjunctive complex expression in Q19. The general strategy is to take the union of the individual table predicates appearing in the disjunctive condition, and filter on this in the respective scan. A further optimization is to rewrite the `NATION` scans as subqueries in the FROM clause:

```
(SELECT (CASE n_name = '[NATION1]' THEN 1 ELSE 0 END) AS nation1,
        (CASE n_name = '[NATION2]' THEN 1 ELSE 0 END) AS nation2
 FROM nation WHERE n_name = '[NATION1]' or n_name = '[NATION2]') cn
```

And subsequently test the join condition as:

```
(sn.nation1=1 AND cn.nation2=1) OR (sn.nation2=1 AND cn.nation1=1)
```

The rationale for the above is that integer tests (executed on the large join result) are faster than string equality. A rewrite like this may not be needed for (column store) systems that use *compressed execution*, i.e. the ability to execute certain predicates in certain operators without decompressing data [8].

CP4.2c: Large IN Clauses

In Q19, Q16 and Q22 (and also Q12) there are IN predicates against a series of at most eight constant values – though in practice OLAP tools that generate SQL queries often create much bigger IN clauses. A naive way to implement IN is to map it into a nested disjunctive expression; however this tends to work well with only a handful of values. In case of many values, performance can typically be won by creating an on-the-fly hash-table, turning the predicate into a semi-join. This effect where joins turn into selections can also be viewed as a "invisible join" [8].

CP4.2d: Evaluation Order in Conjunctions and Disjunctions

In Q19 in particular, but in multiple other queries (e.g. Q6) we see the challenge of finding the optimal evaluation order for complex boolean expressions consisting of conjunctions and disjunctions. Conjuctions can use eager evaluation, i.e. in case of `(X and Y)` refrain from computing expression `Y` if `X=false`. As such, an optimizer should rewrite such expressions into `Y and X` in case `X` is estimated to be less selective than `Y` – this problem can be generalized to arbitrarily complex boolean expressions [31]. Estimating the selectivities of the various boolean expressions may be difficult due to incomplete statistics or correlations. Also, features like

range-partitioning (and partition pruning) may interact with the actually experienced selectivities – and in fact selectivities might change during query execution. For instance, in a `LINEITEM` table that is stored in a clustered index on `l_shipdate`, a range-predicate on `l_receiptdate` typically first experiences a selection percentage of zero, which at some point starts to rise linearly, until it reaches 100% before again linearly dropping off to zero. Therefore, there is an opportunity for dynamic, run-time schemes of determining and changing the evaluation order of boolean expressions [28]. In the case of VectorWise a 20% performance improvement was realized in Q19 by making the boolean expression operator sensitive to the observed selectivity in conjunctions, swapping left for right if the second expression is more selective regularly at run-time – and OR(x,y) being similarly optimized by rewriting it to (NOT(AND(NOT(x),NOT(y))) followed by pushing down the inner NOTs (such that NOT($a > 2$) becomes $a \leq 2$).[2]

CP4.3a: Rewrite LIKE(X%) into Range Query

Q2,9,13,14,16,20 contain expensive LIKE predicates; typically, string manipulations are much more costly than numerical calculations; and in Q13 it also involves `l_comment`, a single column that represents 33% of the entire TPC-H data volume (in VectorWise). LIKE processing has not achieved much research attention; however relying on regular expression libraries, that interpret the LIKE pattern string (assuming the pattern is a constant) is typically not very efficient. A better strategy is to have the optimizer analyze the constant pattern. A special case, is prefix search (`LIKE('xxx%')` that occurs in Q14,16,20; which can be prefiltered by a less expensive string range comparison (`BETWEEN 'xxx' AND 'xxy'`).

CP4.3b: Raw String Matching Performance

The x86 instruction set has been extended with SSE4.2 primitives that provide rather a-typical functionality: they encode 16-byte at-a-time string comparisons in a single SIMD instruction. Using such primitives can strongly speed up long string comparisons; going through 16 bytes in 4 cycles on e.g. the Nehalem core (this is 20 times faster than a normal `strcmp`). However, using these primitives is not always faster, as very short string comparisons that break off at the first or second byte can be better done iteratively. Note that if string comparisons are done during group-by, as part of a hash-table lookup, they typically find an equal string and therefore have to go through it fully, such that the SSE4.2 implementation is best. In contrast, string comparisons done as part of a selection predicate might more often fall in the case where the strings are not equal, favoring the iterative approach.

CP4.3c: Regular Expression Compilation

Complex LIKE expression should best not be handled in an interpretative way, assuming that the LIKE search pattern is a constant string. The database query compiler could compile a Finite State Automaton (FSA) for recognizing the pattern. Another approach is to decompose the LIKE expression into a series of simpler functions, e.g. one that searches forward in a string and returns the new matching offset. This should be used in an iterative way, taking into account backtracking after a failed search.

B.2.5 CP5 CorrelatedSubqueries

CP5.1: Flattening Subqueries

Many TPC-H queries have correlated subqueries. All of these query plans can be flattened, such that the correlated subquery is handled using an equi-join, outer-join or anti-join [41]. In Q21, for instance, there is an EXISTS clause (for orders with more than one supplier) and a NOT EXISTS clause (looking for an item that was received too late). To execute Q21 well, systems need to flatten both subqueries, the first into an equi-join

[2] Swapping the evaluation should only be done if the expression is guaranteed not to trigger run-time errors nor contains NULLs – if not, query behavior could be altered.

plan, the second into an anti-join plan. Therefore, the execution layer of the database system will benefit from implementing these extended join variants.

The ill effects of repetitive tuple-at-a-time subquery execution can also be mitigated in execution systems that use vectorized, or block-wise execution, allowing to run sub-queries with thousands of input parameters instead of one. The ability to look up many keys in an index in one API call, creates the opportunity to benefit from physical locality, if lookup keys exhibit some clustering.

CP5.2: Moving Predicates into a Subquery

Q2 shows a frequent pattern: a correlated subquery which computes an aggregate that is subsequently used in a selection predicate of a similarly looking outer query ("select the minimum cost part supplier for a certain part"). Here the outer query has additional restrictions (on part type and size) that are not present in the correlated subquery, but should be propagated to it. Similar opportunities are found in Q17, and Q20.

CP5.3: Overlap between Outer- and Subquery

In Q2,11,15,17 and Q20 the correlated subquery and the outer query have the same joins and selections. In this case, a non-tree, rather DAG-shaped query plan [37] would allow to execute the common parts just once, providing the intermediate result stream to both the outer query and correlated subquery, which higher up in the query plan are joined together (using normal query decorrelation rewrites). As such, TPC-H rewards systems where the optimizer can detect this and where the execution engine sports an operator that can buffer intermediate results and provide them to multiple parent operators. In Q17, decorrelation, selective join push-down, and re-use together result in a speedup of a factor 500 in HyPer.

B.2.6 CP6: Parallelism and Concurrency

The TPC-H workload consists of two tests: the Power test and the Throughput test. The full query set of the former consists of the 22 TPC-H queries plus two *refresh queries*, which contain both inserts and deletes to the `ORDERS` and `LINEITEM` tables, that delete scattered ranges of orders from the `orderkey` space. In the Throughput test, a number of concurrent Power *query streams*, with different selection parameters, are posed to the system. The implementer can decide in the Throughput run whether to run the refresh streams in parallel with the query streams or not. For the Power test, the geometric mean of all queries results in a Power score. Using the geometric mean implies that the relative improvements to the performance of any query counts equally in the score, regardless whether this is a long-running or short-running query. The upside of this is, is that even as hardware evolves and potentially favors the performance of one query over the other, it remains interesting to optimize the full workload. On the flip-side, one can maintain that for end-users it would normally be more relevant if the long-running queries get optimized. This aspect, absolute run-time, does form part of TPC-H in the form of the Throughput score, which is derived from the full time span it takes to finish all the streams.

CP6.1: Query Plan Parallelization

When TPC-D was conceived, high-end servers would be equipped with a handful of single-core CPU chips (SMP), but very often servers would sport just a single CPU. By 2013, even single-server systems can contain 64 cores; and as such the importance of parallel query performance has increased. In the first decade of TPC-H this only affected the Power test, since it runs every query sequentially, hence it is important that the work gets divided over all cores. With only a few cores available, the Throughput test, which runs 5 (100GB) or 7 (1TB) or more query sets concurrently, could simply run sequential plans on every core and still achieve good system utilization. In the past years, however, having well-performing parallelism is important both in the Power and Throughput tests.

Query plan parallelization in the multi-core area is currently an open issue. At the time of this writing, there is active academic debate on how to parallelize the join operator on many-core architectures, with multiple sophisticated algorithms being devised and tested [9]. We can assume that the current generation of industrial

systems runs less-sophisticated algorithms, and presumably in the current state may not scale linearly on many-core architectures. As such, many-core query parallelization, both in terms of query optimization and query execution is an unresolved choke point.

Further, MPP database systems from the very start focused on scaling out; typically relying on table partitioning over multiple nodes in a cluster. Table partitioning is specifically useful here in order to achieve data locality; such that queries executing in the cluster find much of the data being operated on on the local node already, without need for communication. Even in the presence of high-throughput (e.g. Infiniband) network hardware, communication bandwidth can easily become a bottleneck. For CP6, we acknowledge that the query impact color-classification in Table B.1 is debatable. This classification assumes co-partitioning of `ORDERS` and `LINEITEM` to classify queries using these as medium hard or hard (if other tables are involved as well). Single-table queries parallelize trivially and are white. The idea is that with table partitioning, good parallel speedup is achievable, whereas without it this is harder. Typically, single-server multi-core parallelism does not rely on table partitioning, though it could.

CP6.2: Workload Management

Another important aspect in handling the Throughput test is workload management, which concerns providing multiple concurrent queries as they arrive, and while they run, with computational resources (RAM, cores, I/O bandwidth). The problem here is that the database system has no advance knowledge of the workload to come, hence it must adapt on-the-fly. Decisions that might seem optimal at the time of arrival of a query, might lead to avoidable thrashing if suddenly additional resource-intensive queries arrive. In the case of the Power test, workload management is trivial: it is relatively easy to devise an algorithm that while observing that maximally one query is active, assigns all resources to it. For the Throughput run, as more queries arrive, progressively less resources have to be given to the queries, until the point where there are that many queries in the system and each query gets only a single core. Since parallelism never achieves perfect scalability, in such cases of high load overall, the highest throughput tends to be achieved by running sequential plans in parallel. Workload management is even more complicated in MPP systems, since a decision needs to be made on (i) which nodes to involve in each query and (ii) how many resources per node to use.

CP6.3: Result Re-use

A final observation on the Throughput test is that with a high number of streams, i.e. beyond 20, a significant amount of identical queries emerge in the resulting workload. The reason is that certain parameters, as generated by the TPC-H workload generator, have only a limited amount of parameters bindings (e.g. there are at most 5 different values for region name `r_name`). This weakness opens up the possibility of using a *query result cache*, to eliminate the repetitive part of the workload. A further opportunity that detects even more overlap is the work on recycling [34], which does not only cache final query results, but also intermediate query results of "high worth". Here, worth is a combination of partial-query result size, partial-query evaluation cost, and observed (or estimated) frequency of the partial-query in the workload. It is understood in the rules of TPC-H, though, that any form of result caching should not depend on explicit DBMS configuration parameters, but reflect the default behavior of the system, in order to be admissible. This rule precludes designing re-use strategies that particularly target TPC-H, rather, such strategies should benefit most of the workloads for which the system was designed.

B.3 Conclusions

In this analysis we have (shortly) introduced the concept of "choke points" as being the (hidden) challenges that underlie a benchmark design with the potential to stimulate technological progress. These choke points should point into relevant directions where technological advances are needed; the idea being that the benchmark gives DBMS designers a tangible reward in pursuing solutions for these. We have shown that TPC-H contains a rich set of such choke points, many of which have led to advances in the state-of-the-art in analytical relational database

products in the past two decades; and in fact still contains a number of unsolved challenges. Even despite its age, and arguably reduced value today, we thus argue that TPC-H as introduced in the 1990s (as TPC-D) should be an example for future benchmark designers.