MTD-DS: an SLA-aware Decision Support Benchmark for Multi-tenant Parallel DBMSs

Shaoyi Yin, Franck Morvan, Jorge Martinez-Gil, Abdelkader Hameurlain

Research Report: IRIT/RR--2023--05--FR Nov. 2023

MTD-DS: an SLA-aware Decision Support Benchmark for Multi-tenant Parallel DBMSs

Shaoyi Yin
Paul Sabatier University, Toulouse, France
shaoyi.yin@irit.fr

Franck Morvan
Paul Sabatier University, Toulouse, France
franck.morvan@irit.fr

Jorge Martinez-Gil
Software Competence Center Hagenberg GmbH, Hagenberg, Austria
jorge.martinez-gil@scch.at

Abdelkader Hameurlain
Paul Sabatier University, Toulouse, France
abdelkader.hameurlain@irit.fr

ABSTRACT

Multi-tenant DBMSs are used by Cloud providers for their DBaaS (Database-as-a-Service) products. They could be Single-node RDBMSs installed in VMs, SQL-on-Hadoop systems running on a cluster, or parallel RDBMSs with a shared-nothing or shared-disk architecture. From a Cloud provider's point of view, it is interesting to measure these systems' capability of dealing with multi-tenant workloads. From a tenant's point of view, having the above information on different providers could be helpful in choosing the most suitable one (or several for a multi-cloud deployment). In this paper, we present MTD-DS benchmark (with MTD for Multi-Tenant parallel DBMSs and DS for Decision Support), which extends TPC-DS by adding a multi-tenant query workload generator, a performance SLO (Service Level Objective) generator, configurable DBaaS pricing models, and new metrics to measure the potential capability of a multi-tenant parallel DBMS in obtaining the best trade-off between the provider's benefit and the tenants' satisfaction. Example experimental results have been produced to show the relevance and the feasibility of the MTD-DS benchmark.

CCS CONCEPTS

• Information systems→Data management systems→Database management system engines→Parallel and distributed DBMSs • Information systems→Data management systems→Database management system engines→Online analytical processing engines.

KEYWORDS

Database management systems; Decision support; Benchmark; Multi-tenancy; Service level agreement.

The source code is available online:
https://github.com/shyin2021/MTDDS/

1 INTRODUCTION

More and more multi-tenant DBMSs have been deployed in the Cloud in form of DBaaS (Database-as-a-Service), such as Amazon Redshift [2], Google BigQuery [8], MS Azure SQL [13], Oracle Cloud [20], Snowflake [23], Cloudera Data Warehouse [3], ScaleGrid [22], and many others [6]. The underlying implementations vary and they fall mainly into three categories: Single-node RDBMSs installed in VMs, SQL-on-Hadoop systems (e.g., HIVE, Spark SQL) running on a cluster, and parallel RDBMSs with a shared-nothing or shared-disk architecture. These Cloud-based systems make it possible to develop new decision-support applications very quickly [14]. Moreover, they allow the tenants to reduce dramatically their monetary costs compared to on-premise solutions that run on dedicated servers [14].

From a Cloud provider's point of view, it is interesting to measure its system's capability of dealing with multi-tenant workloads which are characterized by: (1) unstable query arrival rates, (2) different levels of SLA (Service Level Agreement), and (3) heterogeneity of tenants' requirements (in terms of database size, query complexity…). From a tenant's point of view, having the above information on different providers could help in choosing the most suitable one (or several for a multi-cloud deployment). Therefore, it is important to have a benchmark that helps both Cloud providers and tenants in making their choices.

There exist several TPC (Transaction Processing Performance Council) benchmarks in the decision support domain and the most popular one is TPC-DS [16][17][18]. It uses multiple snowflake schemas containing 24 relations. There are 99 SQL queries that cover ad-hoc, reporting, iterative OLAP and data mining type workloads. TPC-DS has already been used directly to study the behavior of SQL-on-Hadoop systems [1][4]. However, since many cloud-related factors, such as the multi-tenant workload characteristics and the various possible pricing models, cannot be directly integrated into the original TPC-DS benchmark, the above evaluations are limited to on-premise settings.

In this paper, we present MTD-DS benchmark (with MTD for Multi-Tenant parallel DBMSs and *DS* for Decision Support), which extends TPC-DS by adding: (1) a multi-tenant query workload generator, (2) a performance SLO (Service Level Objective, an important component in an SLA) generator which

defines a performance SLO for each query in the workload, (3) configurable DBaaS pricing models, and (4) new metrics to measure the potential capability of the multi-tenant parallel DBMS in obtaining the best trade-off between the provider's benefit and the tenants' satisfaction. We expect to make our proposal as an evaluation tool for the following use cases: (1) a current DBaaS provider would like to evaluate its system and test new ideas (e.g., new query optimization methods) in order to increase its economic benefit without hurting the tenants' interests, and (2) a future DBaaS provider would like to choose a DBMS product and a corresponding deployment strategy. Our target users also include non-profitable DBaaS providers whose objective is to self-finance the offered services. Governments who hold the data of all public hospitals and federations of multiple financially autonomous organizations are potential examples. In case of a private Cloud deployment (e.g., inside a single organization), the tenants often use the services for free, but pricing models remain interesting: the provider can generate virtual bills such that the tenants will be aware of what they have consumed and thus pay attention to their future utilization.

The rest of the paper is organized as follows. In section 2, we present the related work. In section 3, we explain our SLA-aware benchmark in detail. In section 4, we analyze the experimental evaluation results and highlight the interest of adopting our proposal. Finally, we conclude the paper in section 5.

2 RELATED WORK

With the increasing adoption of relational database management systems for building Decision Support Systems (DSS) in 1990s [10], the TPC council released its first data warehouse benchmark, called TPC-D, in 1994. As said in [16], TPC-D was later effectively broken due to various technical developments of the DBMSs and then replaced by TPC-H (with ad-hoc queries) and TPC-R (with business reporting queries) in 1999. The latter had never attracted much attention and was decommissioned in 2006. In contrast, TPC-H won a great success very soon because it is closer to the requirements of the industrial decision support applications, and it is still challenging enough today to motivate technical innovations of the DBMS manufacturers. Meanwhile, decision support systems evolved a lot in the 2000's by adopting new relational schemas (e.g., star schema) and mixing various types of queries (e.g., ad-hoc, reporting, iterative and extraction queries). All of these industrial needs could not be covered by TPC-H. Therefore, the TPC council developed another generation of decision support benchmark, called TPC-DS, released in 2006 [16]. TPC-DS extended the data model of TPC-H with multiple snowflake schemas, designed 99 queries (instead of 22 in TPC-H) representing different types of business questions, and overcame several deficiencies of TPC-H in order to be more realistic (e.g., using more representative skewed database content, sub-linear scaling of non-fact tables, and ETL-like data maintenance). TPC-DS has been updated twice [18] and it remains very relevant for current decision support systems.

In the 2010's, with the Big Data movement, efforts were made to integrate as many V's (e.g., volume, velocity, variety, …) as possible into decision support benchmarks. In this context, the TPC council released the TPCx-BB benchmark, which came from the BigBench paper [5]. In this benchmark, not only structured data are considered, but also semi-structured and non-structured data are generated and queried. On the one hand, some machine learning and natural language processing queries are added into the benchmark, and on the other hand, only a subset of the TPC-DS queries are conserved (30 of them) to make the benchmark not so heavy. In our paper, rather than targeting all Big Data applications, we focus on how relational parallel DBMSs tackle the multi-tenancy problem when processing decision support queries. In paper [19], the authors addressed similar issues and they considered the unstable query arrival rates by using a hidden Markov model to simulate the transition between busy and quiet periods. They proposed a new metric based on SLAs and included monetary costs in the evaluation results. Since the paper concentrated on defining a metric to measure the elasticity, the authors made several assumptions for simplification, for example: (1) they treat the workload as a whole without distinguishing individual tenants; (2) the SLAs for all queries are generated in the same way so there is no distinction between urgent and ordinary queries or between premium and basic tenants; (3) the pricing models are not formulated, thus we cannot measure how the DBMS's behaviors change according to different pricing policies. In our paper, we would like to pay more attention on these multi-tenancy related issues. In addition, we will propose more intuitive metrics to guide providers and tenants in making their decisions.

During the writing of this paper, we noticed the publication of the Cloud Analytics Benchmark (CAB) [21]. CAB chose the TPC-H as a foundation and enrich it by adding multi-tenancy concerns, based on an analysis of a real-world query trace. In our paper, we choose to extend TPC-DS due to the aforementioned reasons, but this is not the most important difference compared to CAB. Although having similar observations and objectives, we adopt a different approach for query workload generation, consider the performance SLO in a finer way, and propose new benchmark metrics which serve more directly the service providers to improve their systems. CAB integrates "pay-as-you-go" pricing models in evaluations, while we aim to measure a system's ability to "go-as-you-pay" as well. Here, by using "go-as-you-pay", we refer to the system's ability to adjust the resource consumption for a query according to the tenant's expectation on the response time and the negotiated price. Overall speaking, our work is complementary to CAB.

It is worth mentioning that, the TPC council has an add-on specification called TPC-Pricing [25], which is a consistent set of pricing procedures and methodologies for all TPC benchmarks. For example, for TPC-DS, a Price-Performance metric ($/kQphDS@SF) should be measured. This metric considers the global monetary cost of the System Under Test (SUT), but for us, this cost is independent of the SLA satisfaction and it is not necessarily correlated to the provider's economic benefit. Therefore, new metrics could be interesting for multi-tenant parallel DBMS evaluations.

3 MTD-DS BENCHMARK PROPOSAL

In this section, we present the MTD-DS benchmark proposal, which extends the TPC-DS benchmark on several aspects. First, with regard to the query workload, we generate queries for each tenant by considering various tenant specific factors, such as the tenant's priority, database size and queries' complexity. Multiple tenants arrive simultaneously and the aggregation of all arrived

queries constitute the entire query workload which itself is fluctuating and thus very challenging for multi-tenant parallel DBMSs. Second, for each query of each tenant, we propose a Query Completion Time (QCT for short, meaning the time elapsed between the query submission and completion) threshold as a performance SLO, based on a real execution of the query on the tenant's database. Note that it is different from methods used by DBaaS providers to define the performance SLOs with their tenants, because in that case, it often requires an additional module of the DBMS that interacts with the query optimizer's cost model to get reasonable performance estimates [15][27]. For benchmarking purpose, it is necessary to use the same performance SLOs for all SUTs (Systems Under Test) in order to be fair. Since we are allowed to really execute the TPC-DS queries sequentially and register the execution traces, we prefer designing a benchmark-specific performance SLO generator based on these execution traces rather than pure estimations. The third extension is about the DBaaS pricing models that will be used to deduce the economic income and costs. Finally, we present our new metrics and the benchmark execution rules.

3.1 Multi-tenant Query Workload Generator

To simulate the continuously varying query arrival rates, Poggi et al. [19] consider the entire workload as a whole (i.e., without distinguishing individual tenants) and propose to use a hidden Markov model with a series of *n* discrete levels. At a given point in time, the arrival rate is represented by a series of parameters, describing the probability of being in each level (*1* to *n*), along with the probabilities of transitioning to each of the other levels. As for the workload generation, [19] adapted the notion of streams employed in TPC benchmarks, where for a given number of streams *m*, one instance of each query appears in each stream. In our proposal, for simulating query arrival rates, instead of considering directly the queries of all tenants as a whole, we simulate the query arrivals of each tenant separately. The reasons are as follows: (1) tenant specific characteristics (e.g., the tenant's priority, database size, and queries' complexity) can be considered, (2) the number of tenants can be represented as a variable so that the service provider can choose the number and will have a more intuitionistic view of the simulated query workload, and (3) tenant-level metrics (e.g., SLO satisfaction rate for each tenant and the fairness between tenants) can be measured.

Each tenant is modeled as a tuple <*TnID, TnPriority, DBsize, QueriesComplexity, TnArrivalTime, NbQueries, Lambda, IdleRatio, PeakRatio, f_peak*>. Table 1 shows the meaning of these parameters. Their values differ from one tenant to another, and the simple aggregation of all tenants' queries forms naturally the entire query workload (see Table 3). First, we differentiate the tenants according to the priority, the database size and the queries complexity. Three levels of tenant priority will be considered: basic, standard and premium. They differ from each other in terms of performance SLOs and prices of their queries, which will be explained in Section 3.2 and Section 3.3. With regard to the database size, we use three different scale factors to represent small, medium and large databases. As for the query complexity[1], the following workload streams can be generated: (1) simple workload streams, consisting only of the simple queries with less than 4 binary operators; (2) complex workload streams, consisting only of the complex queries with more than 6 binary operators; and (3) mixed workload streams, consisting of all the original TPC-DS benchmark queries. The list of queries in each complexity category can be found in Table 2.

Based on the above characteristics, 27 combinations (with respect to TnPriority, DBSize and QueriesComplexity) could appear in the complete generated workload. In the benchmark, we treat the total number of tenants as a variable whose value will be chosen by the benchmark user. To be more realistic, we suggest that the proportion of standard tenants reaches at least 50%.

Table 1: Parameters used to characterize a tenant

Parameter	Signification
TnID	The tenant's identifier
TnPriority	The priority of the tenant
DBSize	The size of the database used for the tenant (small, medium or large)
QueriesComplexity	The complexity of the tenant's queries (simple, medium or complex)
TnArrivalTime	The arrival time of the tenant's first query
NbQueries	Total number of queries launched by the tenant
Lambda	The expected value of the number of query arrivals per time unit
IdleRatio	The ratio between the number of idle intervals and the total number of time intervals occupied by the tenant
PeakRatio	The ratio between the number of peak intervals and the total number of time intervals occupied by the tenant
f_peak	The multiplication factor for the parameter Lambda during the peak time

Table 2: Simple, medium and complex queries

Query complexity	Query IDs
Simple	3, 12, 15, 20, 21, 22, 28, 36, 37, 41, 42, 43, 52, 53, 55, 63, 80, 81, 82, 86, 89, 93, 96, 97, 98 (25 queries)
Medium	1, 2, 5, 6, 7, 10, 16, 19, 26, 27, 30, 32, 34, 35, 38, 40, 45, 47, 50, 51, 57, 62, 67, 68, 70, 71, 73, 79, 84, 90, 91, 92, 94, 95, 99 (35 queries)
Complex	4, 8, 9, 11, 13, 14, 17, 18, 23, 24, 25, 29, 31, 33, 39, 44, 46, 48, 49, 54, 56, 58, 59, 60, 61, 64, 65, 66, 69, 72, 74, 75, 76, 77, 78, 83, 85, 87, 88 (39 queries)

The number of queries launched by each tenant is a random value between 10 and 100. To amplify the instability of the query arrival rates, we distinguish idle, peak and normal periods for each tenant. During an idle period, no query arrives, while during a peak period, queries arrive very frequently. For each normal period, we assume a Poisson distribution for the query arrivals, with *Lambda* the expected value of the number of query arrivals per time unit. More precisely, the arrival times of *N* queries are generated with the following process [9]: First, simulate exponential random variables $X_1, X_2, \ldots, X_{N-1}$ representing

[1] For simplicity, we define the query complexity only based on the number of binary operators (joins, unions, etc.) in the query.

interarrival times, based on the value of *Lambda*. Then, set Y_1 = *TnArrivalTime,* $Y_2 = Y_1 + X_1, \ldots, Y_N = Y_{N-1} + X_{N-1}$, where the variables $Y_1, \ldots, Y_N$, represent the arrival times of the tenant's *N* queries. Note that, during a peak period, the number of queries arrived per time unit is multiplied by a factor *f_peak*. Concerning the distribution of idle, peak and normal periods, we fix a specific ratio for each tenant (e.g., 10% idle time, 10% peak time and 80% normal time for tenant *T*). The tenant arrival time itself is generated in the way that the tenant arrivals also follow a Poisson distribution, with 3 tenant arrivals per time unit in average.

In Table 3, we show the first six tenants of the generated workload. Figure 1 shows the query arrivals per time unit of 20 tenants, and Figure 2 gives the aggregative query arrival trend for the complete query workload. Finally, in Figure 3, we focus on three representative tenants: Tenant 4 has more idle periods (40%) than peak periods (12%), Tenant 8 has more peak periods (49%) than idle periods (12%), while Tenant 16 differs less in peak (54%) and idle periods (37%).

Table 3: Examples of tenants' information

TnID	TnPriority	DB Size	Queries Complexity	TnArrivalTime	NbQueries	Lambda	IdleRatio	PeakRatio	f_peak
1	Standard	Medium	Simple	0	26	8	29	23	2
2	Basic	Small	Mixed	13.58	73	8	39	53	3
3	Basic	Medium	Simple	20.76	38	6	48	13	2
4	Premium	Small	Complex	25.01	48	2	40	12	2
5	Standard	Large	Simple	27.32	36	3	14	36	2
6	Standard	Medium	Mixed	28.17	69	6	29	47	2

3.2 Performance SLO Generator

Most of the commercial DBaaS providers do not define performance SLOs for individual queries yet, and the bills received by the tenants are independent of QCTs (different billing policies will be discussed in section 3.3). Nevertheless, many tenants expect strongly to have some performance guarantees along with a pricing policy which pushes the provider to meet the performance SLOs as much as possible. Some existing research work [15] [27] proposes different ways to define personalized SLAs for DBaaS tenants. We believe that including performance SLOs in an SLA is a reasonable requirement and will be realized in the near future, even though many efforts still need to be made in this direction, as we will show later on in detail. In fact, to some extent, this requires predictable performance of the system, which has been recognized to be a major research challenge for the serverless computing paradigm to come true [24].

In this paper, our focus is not to propose another method for defining SLAs, but to evaluate multi-tenant parallel DBMSs by assuming that performance SLOs will exist and they should consequently have an economic impact. For evaluation and comparison purposes, in order to be fair, we design a performance SLO generator which provides a QCT threshold for each benchmark query, and this threshold should be considered by the SUT, because a penalty needs to be paid in case of violation. For fixing this threshold, we distinguish two types of queries: ad-hoc queries and reporting queries. The former expects an immediate response and the latter has a fixed deadline that is not necessarily urgent but often rigid. For ad-hoc queries, we assume that the tenant has an expected QCT. However, the tenant could tolerate some delay if the price is reduced accordingly and each priority level has its own tolerance rate. Therefore, as in [12], we introduce a factor τ (Tolerance rate threshold) such that a penalty is only paid if the real QCT of the query *q* running on a database of scale factor *sf* is larger than *τ* times of the QCT threshold in the SLO, i.e., $QCT_{q,sf} > \tau * QCT\_SLO_{q,sf}$. As for the reporting queries, a hard deadline is defined, which is a precise clock time. A penalty should be paid in case of any delay.

In TPC-DS, ad-hoc queries are those concerning the Store and Web sales, while reporting queries are the remaining ones which concern the Catalog channel[2]. In our proposal, it is logical to distinguish both types of queries in the same way. To generate the performance SLOs, we first run the TPC-DS queries in the Power Test mode (i.e., running the queries sequentially without concurrent sessions) under a System of Reference (SOR) on a database of scale factor SF_{SMALL}, SF_{MEDIIM}, and SF_{LARGE} respectively and register the QCT of each query with each scale factor. Here, an SOR is defined as a multi-tenant parallel DBMS installed on top of a hardware infrastructure by the provider with an initial configuration. For example, in our experimental demonstration, we use a parallel version of PostgreSQL running on a cluster of 5 virtual machines. Then, SF_{SMALL}, SF_{MEDIUM}, and SF_{LARGE} correspond to the possible values of the *DBSize* parameter of the simulated tenants. In fact, both the SOR and the three *SF* values depend on the business scale of the provider. Each user of the benchmark should decide them according to the actual or potential needs of their tenants.

Based on the obtained numbers through the Power Tests, we define the *QCT_SLO* for a query *q* running on a database of scale factor *sf* by Equation [1], where $\tau_{REPORTING}$ needs to be fixed before running the benchmark.

$$QCT\_SLO_{q,sf} = \begin{cases} QCT_{q,sf}, & if\ q\ is\ an\ ad-hoc\ query \\ QCT_{q,sf} * \tau_{REPORTING}, & if\ q\ is\ a\ reporting\ query \end{cases} \quad [1]$$

In the case of comparing different SUTs, we generate a set of *QCT_SLO* for each SUT separately and then apply the following principles for each query: (1) in an isolated environment (i.e., single tenant, fixed amount of resources), at least one SUT can meet the SLO strictly, without any delay; and (2) there should exist situations in which the SLO is not met strictly but within the tolerance threshold (i.e., the QCT is between $QCT\_SLO_{q,sf}$ *and* $\tau_{min} * QCT\_SLO_{q,sf}$), such that the performance differences between systems could be observed. Here, the value of τ_{min} is the

[2] We recall that the TPC-DS imitates the business activity of a large retail company and tracks its store, web and catalog sales channels.

tolerance rate threshold of the tenants with the highest priority (i.e., Premium). Equation [2] defines the final *QCT_SLO*, in which *X* represents the set of SUTs to compare.

$$QCT\_SLO_{q,sf} = max(\max_{x \in X}\left(\frac{QCT_{SLO_{q,sf,x}}}{\tau_{min}}\right), \min_{x \in X}(QCT_{SLO_{q,sf,x}})) \quad [2]$$

3.3 Configurable Pricing Models

The pricing models used by commercial DBaaS providers fall into two categories. In the first category, dedicated resources can be allocated to a tenant for executing queries and the price depends on the resource amount and the query duration. In the second category, the price of a query is based on the input data size. Academic researchers believe that the performance SLOs should be included in the SLA and the bills for the tenants should be computed based on the real performance with respect to the SLO satisfaction[15][27]. Therefore, we introduce another two pricing models. All these models will be integrated into the benchmark, in order that the provider could instantiate (and enrich) one of them according to its business requirements. The details of each pricing model are as follows.

(1) Resource Consumption-Based pricing model

With the Resource Consumption-Based (RCB) pricing model, the provider fixes a price for each resource (e.g., CPU, RAM, disk or network). Very often, a certain amount of CPU, RAM and disk space is capsulated in a physical or virtual machine, whose price is based on the duration of utilization, in form of money per time unit (e.g., hour). When the network usage is billed, the price is computed based on the volume of data transferred during query executions, in form of money per unit of data volume (e.g., megabytes).

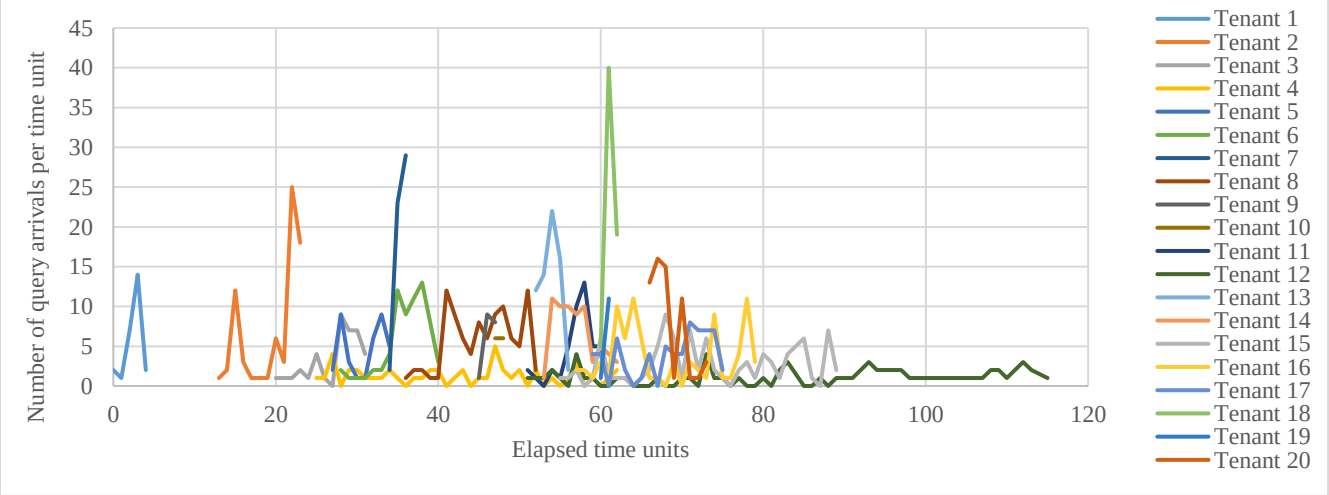


Figure 1: Number of query arrivals per time unit of each tenant

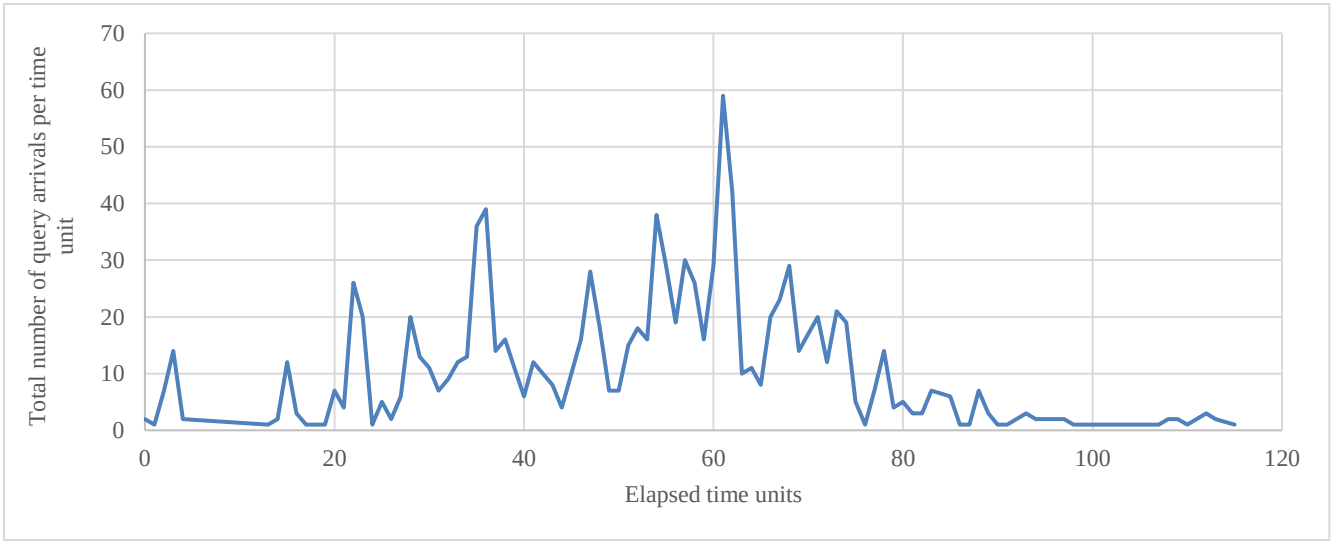


Figure 2: Total number of query arrivals per time unit

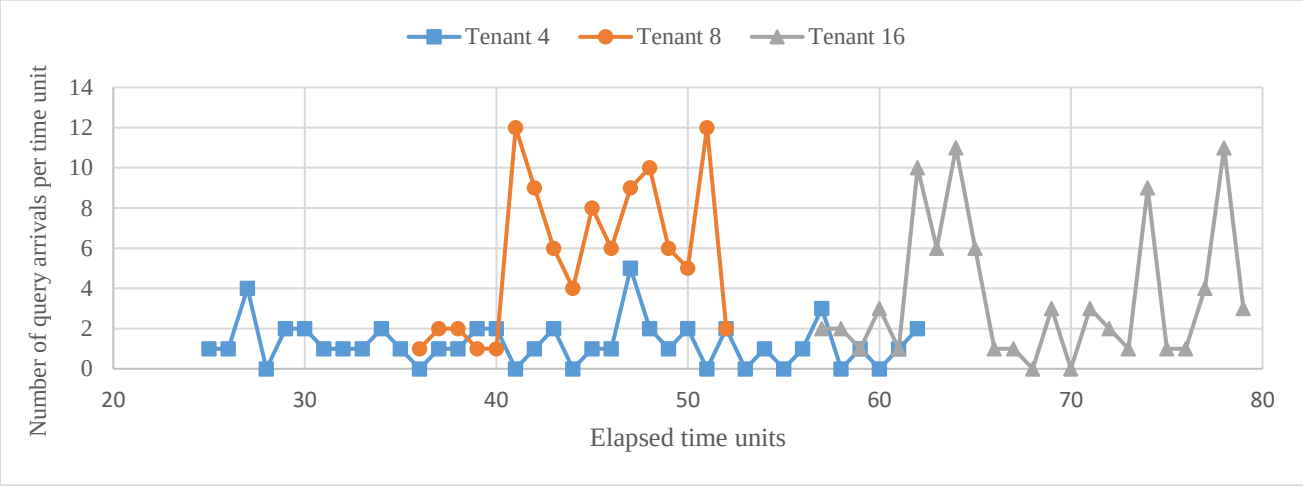


Figure 3: Number of query arrivals per time unit of three typical tenants

(2) Input Data Size-Based pricing model

With the Input Data Size-based (IDS) pricing model, the price of a query depends on the size of the input data, in form of money per unit of data volume (e.g., megabytes). The latter can be calculated precisely, so the invoice is fully auditable for the tenant. With this model, the tenant priority is not considered in the billing process.

(3) Service Tier-Based pricing model

With the Service Tier-based (STB) pricing model, several service tiers are proposed to the tenant [15]. In each tier, there are multiple query types (e.g., simple, complex, medium) and for each type, one performance SLO in terms of QCT threshold is defined. The price of a service tier is in form of money per time unit (e.g., hour). If a tier proposes a lower QCT threshold, normally it should consume more resources to meet the SLO and correspondingly, the price announced for this tier is higher. The tenant chooses one tier according to the applications' requirements. When there is an SLO violation, the provider pays a penalty to the tenant.

(4) Query Level SLO-Aware pricing model

With theQuery Level SLO-Aware (QLSA) pricing model, the price of each query depends on the performance SLO satisfaction. The tolerance rate threshold (the factor τ), which varies from one tenant to another, is also considered. Normally, a tenant with a higher priority are less tolerant to delays, but are ready to pay more money when the SLO is met. If the SLO is met strictly, the tenant pays the full price. If the SLO is not met strictly but in the tolerance delay, the tenant pays partially the initial price. Otherwise, the SLO is violated and the provider should pay a penalty. One possible way is to calculate the final price with a pricing function (Equation [3]) included in the SLA [27]:

$$PR_{SLA} = \begin{cases} PR_{exp}, if QCT \le QCT_{exp} \\ \left(\frac{QCT_{exp}}{QCT}\right) \times PR_{exp}, if QCT_{exp} < QCT \le \tau \times QCT_{exp} \\ -PR_{exp}, if QCT > \tau \times QCT_{exp} \end{cases} \quad [3]$$

Where *QCT* is the real query completion time and QCT_{EXP} is the expected query completion time, which has the same meaning as *QCT_SLO* previously defined in this paper. PR_{EXP} is the expected price to pay if the SLO is strictly met, and it is supposed to be calculated by using the query optimizer according to the amount of resources that would be consumed and the tenant's priority. In this benchmark, we assume that the resources are measured in terms of number of nodes. In a shared-nothing cluster, a node contains a processor, a certain amount of memory and some external storage space. For other parallel architectures, the notion of node should be adjusted. Precisely, the expected price of a query *q* for a tenant *tn* is generated by using Equation [4].

$$PR_{EXP}(q, tn) = Nb\_nd(q, tn) * Price\_per\_nd * QCT_{EXP} * \tau_{max} / \tau(tn) \quad [4]$$

Where *Nb_nd(q, tn)* is the number of nodes that are assumed to use for executing the query *q* of the tenant *tn*. *τ(tn)* is the tolerance rate threshold of the tenant *tn*. When a tenant's tolerance rate threshold is smaller, it is considered as someone with higher priority, because that tenant's queries are usually more urgent. Consequently, more money should be paid if the urgency is correctly considered. In Equation [4], τ_{max} represents the tolerance rate threshold of the tenants with the lowest priority (i.e., Basic). Then $\tau_{max} / \tau(tn)$ determines the multiplication factor to compute the query price of the tenant *tn*.

3.4 Cost-effectiveness Metrics

The proposed benchmark is assumed to be used by a DBaaS provider to choose a multi-tenant parallel DBMS among several candidates, or to validate a new version of an existing one. We define tenant-oriented metrics to assure that the service is appealing enough for tenants, as well as provider-oriented metrics for the provider's internal use.

(1) Tenant-oriented metrics

The DBaaS provider should be interested in the tenants' satisfaction. Only when the service is appealing to tenants, it will make sense to improve the provider-oriented metrics. Therefore, we propose three Tenant-Oriented metrics (*TO metrics* for short) which serve as estimators of tenant feedback. We measure them in the benchmarking process, on behalf of the provider, to estimate the tenants' satisfaction after using the service. Later on, when the system is served by real tenants, the values of the first two metrics

could be obtained statistically and become QoS indicators, while the value of the third one has no more sense. This could be understood easily after reading the definition of the three metrics below.

TO_Metric 1: SSR (SLO Satisfaction Rate), which is the percentage of the number of queries that have been successfully executed over the total number of queries. This calculation applies to the entire workload (Equation [5]).

$$SSR = \frac{Number\ of\ executed\ queries\ meeting\ the\ SLO}{Total\ number\ of\ submitted\ queries} \quad [5]$$

The metric *SSR* is straight-forward but we choose it not only for its simplicity. The reason of considering each query equally regardless of its complexity, urgency and duration is that, these factors are already considered when defining the SLO, the price and the potential penalty of the query. Therefore, the rejection of any query represents the same level of loss for all tenants. In fact, when the tenant considers that one query is more "important" than another, a higher price will often be paid for that query, and in case of SLO violation, the refund is also higher. Thus, there is no need to add tenant or query level co-efficiencies when we measure the tenants' global satisfaction.

TO_Metric 2: FTS (Fairness between the Tenants' Satisfaction rates). For the same query, a tenant with a lower priority is ready to pay less money but also ready to have a longer delay to get the query result. The provider can guide the tenant to choose the most appropriate priority level, but once the performance SLOs are established, the provider should be responsible for meeting them. Therefore, tenants with a low priority should be treated equally and should have the same level of satisfaction with regard to what is expected. To measure that, we compute the SLO satisfaction rate of each tenant and then calculate the standard deviation, as shown by Equation [6], where n is the number of tenants chosen for the benchmarking process, SSR_i is the SLO satisfaction rate of the tenant i and $\overline{SSR}$ is the average *SSR* of all tenants. If the standard deviation is 0, meaning that all *SSR*s are the same, we assign the *FTS* score to be 1. Otherwise, we assign the *FTS* score to be the inverse of the standard deviation.

$$FTS = \begin{cases} 1,\ if\ SSR_i = SSR_j, \forall\ i \neq j \\ 1/\sqrt{\frac{1}{n}\sum_i^n (SSR_i - \overline{SSR})^2},\ otherwise \end{cases} \quad [6]$$

TO_Metric 3: TPAT (Total Payment of All Tenants), which is simply the sum of the bills of all tenants. This metric is even more straight-forward and has been naturally used by most of the benchmarks cited in Section 2. We cannot ignore it: the query workload is the same for all SUTs, and therefore, the difference on the final bills has a significant meaning for the tenants, especially when there are similar results on the other metrics. The added-value of this metric in the MTD-DS benchmark is that, this metric is measured by using the pricing model of the DBaaS provider that is explicitly described and integrated into the benchmark. When the same pricing model is used for both SUT1 and SUT2, if other metrics show similar scores for both systems, but the TPAT of the SUT1 is much higher than that of the SUT2, we could doubt about the capacity of the SUT1 (e.g., the resource utilization is suboptimal). In other benchmarks, tests are often made on two DBaaS providers by treating them as black-boxes. In that case, when the bill of one provider is higher, we cannot guess it is due to the system capacity or simply caused by the pricing policy.

(2) Provider-oriented metrics

The main objective of a DBaaS provider is to maximize its long-term economic benefit. However, simply compute a final gain is not enough to show that a multi-tenant parallel DBMS adapts well to query workload variations and resource limitations. In other words, if a system allows a high benefit for a fixed query workload by using a fixed amount of resources, we cannot deduce if it can bear changing workloads without losing money. Therefore, we first define two Provider-Oriented metrics (*PO metrics* for short) that need to be measured by stressing the system progressively, representing two kinds of saturation thresholds:

PO_Metric 1: MNN (Minimum Number of Nodes required before a system saturation) under the reference query arrival rate.

PO_Metric 1 Bis: HARF (Highest Arrival Rate Factor before a system saturation) under the reference number of nodes.

A ***system saturation*** stands for the situation where the LUBF (Long-term Unit Benefit Factor, Equation [7]) becomes negative, due to excessive SLO violations, or the SLO satisfaction rate (SSR defined above) is lower than a certain value (SSR_{MIN}).

LUBF = Total economic benefit / Total elapsed time [7]

In [8], the total economic benefit is computed by removing the resource consumption costs and the penalties from the sum of the income brought by all queries. The total elapsed time is the time elapsed from the arrival of the first query and the completion of the last query of the complete query workload.

In the definition of these two metrics, the reference query arrival rate corresponds to an initial target query arrival rate that the system is designed for, while the reference number of nodes corresponds to the number of nodes that the DBaaS provider would like to start with. Very often, only one of these two values can be imposed by the provider when performing the benchmark, and only one of the two metrics (MNN and HARF) is of interest.

In addition to MNN and HARF which test the behaviors of the SUT under pressure, we also propose another two PO metrics which indicates the potential of the SUT to maximize the provider's long-term benefit.

PO_Metric 2: ONN (Optimal Number of Nodes) which gives the highest LUBF under the reference query arrival rate (if it is known).

PO_Metric 2 Bis: OARF (Optimal Arrival Rate Factor) which gives the highest LUBF under the reference number of nodes (if it is known).

Normally, during a benchmarking process, either MNN and ONN are measured, or HARF and OARF are measured, along with the corresponding LUBFs. Moreover, in case that HARF and OARF are measured, one of them could be more important than the other. For example, for a commercial DBaaS provider, the OARF is more meaningful for it to regulate the number of accepted tenants in order to fit the optimal query arrival rate. In contrast, for a non-profitable DBaaS provider (e.g., a government or a multi-organization federation), the objective is to serve all eligible tenants without loosing money. Therefore, the HARF is a more important indicator in its decision-making process.

(3) All together for comparing SUTs

When there are several SUTs to compare, we normalize the values of all metrics into the interval [ARF_{min}, ARF_{max}] and show them together within a radar chart. For the *ARF* dimension, we preserve the original values. For the *SSR*, *FTS* and *LUBF* dimensions, we divide the original value of each SUT by the maximum one of all SUTs and then multiply with the maximum value of all *ARF*. For the *TPAT* dimension, we divide the minimum value of all SUTs by the original value of each SUT respectively and multiply with the maximum value of all *ARF*. In this case, in each dimension, the best SUT gets the full score.

We expect that, there is at least one SUT that outperforms all others in all dimensions and it will be the "winner". In section 4, we will demonstrate how this could be technically possible, especially when comparing two versions of the same original system. In consequence, the providers are motivated to improve their systems through innovations, which allows obtaining their business objectives without hurting the tenants' interests.

Figure 4 gives an example radar chart for a non-profitable DBaSS provider, where the HARF metric is used and the corresponding LUBF is drawn. In this example, System B outperforms both System A and System C. Therefore, the provider would finally decide to adopt System B for its DBaaS.

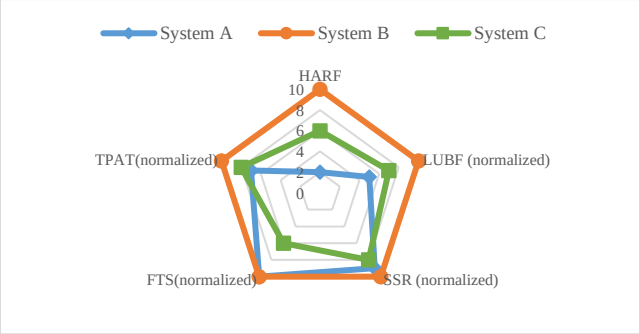


Figure 4. An example radar chart for comparing systems

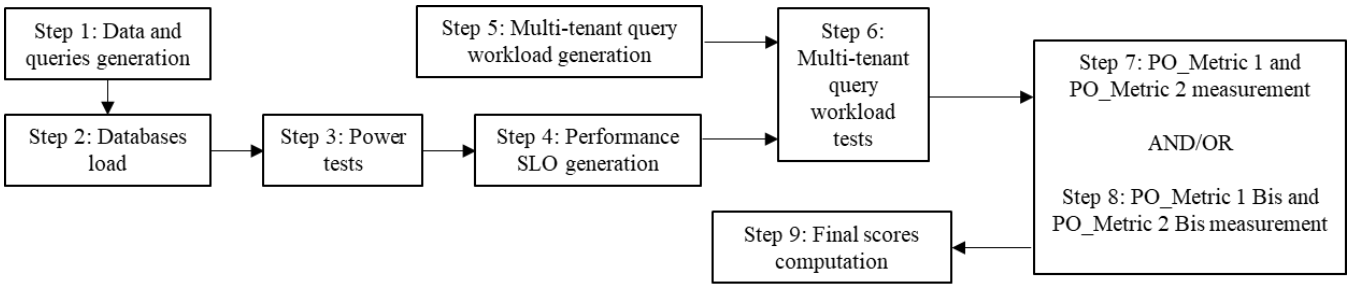


Figure 5: The MTD-DS benchmark execution rules

3.5 Execution Rules

To run the benchmark, we define the steps to follow as below.

Step 1. Use the TPC-DS tool kit to generate data and queries for three scale factors SF_{SMALL}, SF_{MEDIUM}, and SF_{LARGE} respectively. We recall that the exact values of these scale factors are chosen by the service provider according to its estimated or real business scale.

Step 2. Create the three databases and load the generated data into them. Create indexes and auxiliary data structures which could be used by the reporting queries.

Step 3. Run the TPC-DS queries in the *Power Test* mode on each of the three databases and register the QCT of each query for each chosen scale factor.

Step 4. Run the performance SLO generator to define the *QCT_SLO* for each query with respect to each representative scale factor.

Step 5. Run the multi-tenant query workload generator for the chosen number of tenants.

Step 6. Launch the query workload on the SUT, similar to the *Throughput Test* of the TPC-DS. The difference is that, the query arrival times are determined in Step 5 by running the multi-tenant query workload generator. Register the execution traces.

Step 7. Repeat Step 6 by adding or removing one node to the SUT in each iteration until detecting both the MNN and the ONN.

Step 8. Repeat Step 6 by increasing or decreasing one to the Arrival Rate Factor (ARF) in each iteration until detecting both the HARF and the OARF.

Step 9. Based on the results of Step 7 and/or Step 8, calculate the scores of all metrics.

These steps and their execution order are shown in the Figure 5. We summarize the input parameters of the benchmark execution in Table 4.

Table 4: Input parameters of the MTD-DS benchmark

Parameter	Signification
SF_{SMALL}	A representative scale factor of the small databases.
SF_{MEDIUM}	A representative scale factor of the medium size databases.
SF_{LARGE}	A representative scale factor of the large databases.
NbTenants	The number of tenants to generate for the benchmarking.
τ_{BASIC}	The tolerance rate threshold of the "basic" tenants.
$\tau_{STANDARD}$	The tolerance rate threshold of the "standard" tenants.
$\tau_{PREMIUM}$	The tolerance rate threshold of the "premium" tenants.
$\tau_{REPORTING}$	The tolerance rate threshold used to generate the deadline of a reporting query.

4 EXAMPLE EXPERIMENTAL RESULTS

In this section, we present some preliminary experiments and their results in order to: (1) illustrate how the benchmark could be used; (2) show the feasibility of the benchmark, i.e., its ability to run on a real system within a reasonable benchmarking duration; and (3) validate the relevance of the proposed metrics, i.e., to see if the results can be easily interpreted to draw valuable insights. First, we describe the experimental environment. Then, we take two pricing models as examples to show some obtained results.

4.1. Experimental Settings

Since the target users of the MTD-DS benchmark are DBaaS providers, it is inadequate for us to use directly existing commercial DBaaS systems, because we can only play the role of a tenant thus do not have any control on the internal implementation or configuration of those systems. It would be preferable to install multi-tenant parallel DBMSs on top of a Cloud-scale cluster (which can be rented from an IaaS provider). However, this is too much time and money consuming with regard to our objective (i.e., to demonstrate the benchmarking process). Therefore, we chose to configure a tiny but real cluster with our own materials for a DBaaS deployment. A brief description of the experimental settings is given below.

(1) Multi-tenant parallel DBMS configuration

We use Postgres-XL[3], a parallel version of PostgreSQL, as the underlying multi-tenant parallel DBMS of the SUTs. It is installed on top of 5 virtual machines residing in 2 physical personal computers connected by a local network. The virtual machines have their own IP addresses and can access each other by using the SSH (Secure Shell) protocol. The main characteristics of these machines can be found in Table 5 and Table 6.

Table 5: Main characteristics of the physical computers

Name	CPU	RAM Size	External Storage	OS
PC1	2.30GHz, 16 cores	16 GB	SSD, 512 GB	Windows 10
PC2	1.70GHz, 8 cores	16 GB	SSD, 256 GB	Windows 10

Table 6: Main characteristics of the virtual machines

Name	Nb. CPU cores	RAM Size	External Storage Size	OS	Physical computer
Master	2	4 GB	120 GB	Ubuntu	PC1
Data1	2	4 GB	60 GB	Ubuntu	PC1
Data2	2	4 GB	60 GB	Ubuntu	PC1
Data3	2	6 GB*	60 GB	Ubuntu	PC2
Data4	2	4 GB	60 GB	Ubuntu	PC2

**The RAM size of Data3 was increased during our experiments because it was discovered to be the busiest node and 4GB was not enough to handle multiple concurrent tenant connections.*

The Postgres-XL cluster contains one GTM (Global Transaction Manager), one coordinator and 4 data nodes. The GTM and the coordinator are collocated in the virtual machine *Master*. Each data node uses a separate virtual machine. The global architecture is shown in Figure 6. We used the default values for most PostgreSQL configuration parameters, except the ones listed in Table 7.

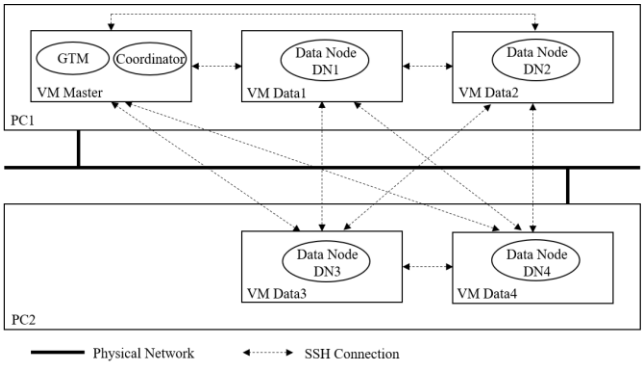


Figure 6: Postgres-XL cluster deployment architecture

Table 7: Customized PostgreSQL configuration parameter values[4]

Parameter	Customized value
shared_buffers	512 MB
temp_buffers	64 MB
work_mem	32 MB
effective_io_concurrency	10
enable_material	off
enable_nestloop	off (except for queries with "Exists")
random_page_cost	1.1
parallel_setup_cost	100

(2) Benchmark related parameter values

Constrained by the above system deployment and configuration, we tested the MTD-DS benchmark in a relatively small-scale setting, such that a benchmarking run lasts no more than 24 hours. The chosen values could be found in Table 8.

Table 8: Benchmark related parameter values

Parameter	Value used for the experimentation
SF_{SMALL}	1
SF_{MEDIUM}	2
SF_{LARGE}	3*
NbTenants	6*
τ_{BASIC}	100
$\tau_{STANDARD}$	50
$\tau_{PREMIUM}$	10
$\tau_{REPORTING}$	1000

**We were not able to use larger values for these two parameters, because each tenant has its own database and the concurrent execution of queries from multiple tenants requires a lot of RAM, or extremely long execution times caused by heavy swapping.*

(3) Tenants' placement

In fact, the example shown in Table 3 (in Section 3.1) contains the real values produced by the query workload generator. Therefore, we have 2 tenants (Tenant 2 and Tenant 4) with small databases, 3 tenants (Tenants 1, 3 and 6) with medium size databases, and 1 tenant (Tenant 5) with a relatively large database.

[3] Repository: https://git.postgresql.org/gitweb/?p=postgres-xl.git

[4] The signification of these parameters could be found here: https://www.postgresql.org/docs/11/runtime-config.html, mainly in "Resource Consumption" and "Query Planning" sections.

The placement of these tenants is done as follows: (1) the number of nodes for each tenant corresponds to the scale factor of its database; (2) when there are 2 or 3 data nodes needed by a tenant, we take at least one data node from each physical computer; (3) globally, the data nodes are allocated to the tenants of each scale in a round-robin way. The resulting tenant placement matrix is given in Table 9.

Table 9: Tenant placement matrix

	DN1 (on PC1)	DN2 (on PC1)	DN3 (on PC2)	DN4 (on PC2)
Tenant 1	×		×	
Tenant 2				×
Tenant 3		×		×
Tenant 4			×	
Tenant 5	×	×	×	
Tenant 6	×		×	

For each tenant, the fact tables are partitioned by using the hash partitioning method, while the dimension tables are replicated on all nodes allocated to the tenant.

(4) SUTs description

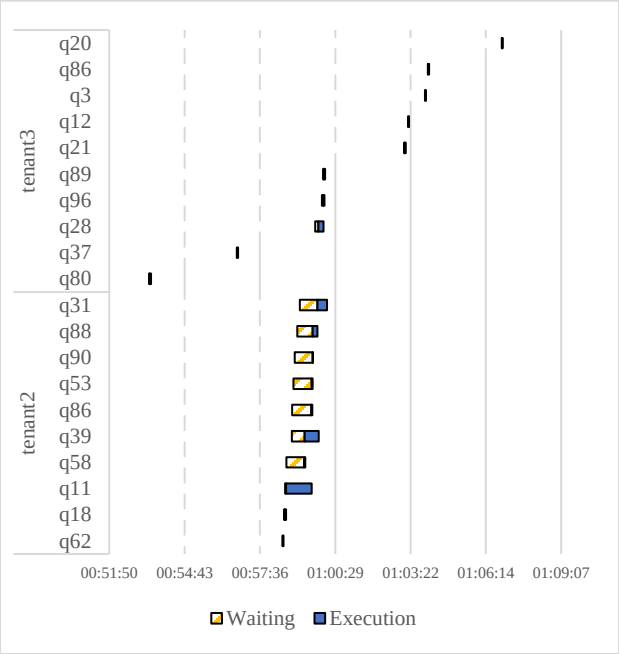
We have implemented a driver program which submits the queries of all tenants to Postgres-XL according to the arrival times defined in the generated query workload. When Postgres-XL is too busy and refuses a query, this query will be queued by the driver and resubmitted later on. Interestingly but not surprisingly, the way how these queries are queued has a big impact on the final benchmarking scores. Based on this observation, we have implemented two different ways to order concurrent queries and treat them as two SUTs (SUT1 and STU2) for comparison. We choose to do this in the driver level rather than modifying the parallel DBMS kernel, because the latter is much more complicated and itself is actually a challenging research topic. We are actually working on that in parallel of the benchmark proposal [7]. In the current paper, we do not seek the optimality but only the diversity when choosing the SUTs. Below, we highlight the differences of the chosen SUTs.

In SUT1, if a query cannot be accepted immediately by Postgres-XL, it is added to the end of the waiting queue. Then, the FIFO (First-In-First-Out) policy is applied. The number of allowed concurrent queries by Postgres-XL is set to 3, due to the resource limitation.

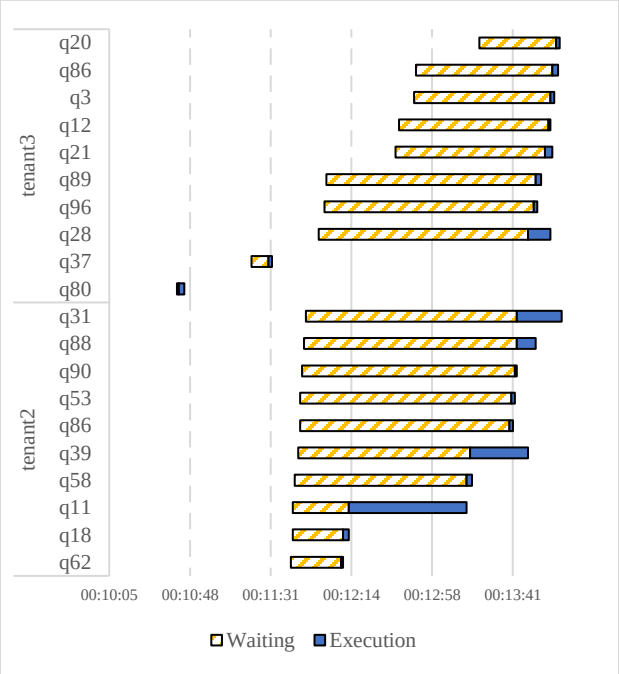
In SUT2, we maintain a second queue for the long-running queries, in order to avoid disproportionally long waiting time of the short-running queries. The number of allowed concurrent queries by Postgres-XL is also set to 3, but at any moment, at most 1 long-running query is handled by Postgres-XL. therefore, 2 or 3 short-running queries could execute concurrently.

4.2. Execution Traces Analysis

For each of the 290 queries in the generated workload, we create a threading timer in the driver program to control the launch time. When a query is accepted by Postgres-XL, it starts executing. When the execution is finished, Postgres-XL returns a success message to the driver program. Therefore, we were able to register the execution traces of all queries in a file according to the following format *<TenantId, QueryId, TimerId, Event, Timestamp>*. The *Event* element corresponds to *Launched, Started* or *Finished*. After executing all queries, we transform the traces into another format that will be used by the billing process and the visualization. In the transformed traces, we have *<SUT_Name, ClusterSize, ArrivalRateFactor, TenantId, QueryId, TimerId, LaunchTime, StartTime, FinishTime>* and we can deduce the *Wating time* and the *Execution duration* of each query.



(a) ARF = 2



(b) ARF = 10

Figure 7. Waiting time and execution duration of each query for the SUT2 under 2 different query arrival rates

In Figure 7, we visualize the execution traces of SUT2 under 2 different query arrival rate factors. For a better readability, we will not show the traces of all queries but only zoom on several queries of Tenant2 and Tenant3. We can observe that, when the query arrival rate factor is high (Figure 7(b)), many queries should wait in the queues before being executed. However, we cannot conclude that it represents a worse scenario than that in Figure 7(a) without considering precisely the performance SLOs. If the tenants do not feel very disappointed regardless of the delay, the provider would prefer the scenario in Figure 7(b), because the resources are fully utilized during the query arrival period. Therefore, carefully chosen metrics are necessary for the provider to adjust its business ambition with regard to the invested resources, or to calibrate the resources with regard to its business scale. Below, we will examine if our proposed metrics could help facilitating the provider's decision making, by instantiating two different pricing models: RCB and QLSA.

4.3. Results with the RCB Pricing Model

We start by using the RCB pricing model, meaning that the tenants' bills are calculated based on the amount and the duration of resource consumption during their query executions, as described in Section 3.3. With this model, we compare SUT1 and SUT2. The global trend is that the LUBF increases when the query arrival rate factor is higher, even though the tenants' satisfaction rate and the fairness factor decrease (see Figure 8 and Figure 9). Therefore, the conflict of interests between the provider and the tenants is obvious. In this case, the threshold for the tenants' SLO satisfaction rate (SSR_{MIN}) should be declared explicitly. In our example, we use SSR_{MIN} = 70%. Based on this threshold, we can deduce that, both the HARF and OARF values for SUT1 is 2, and those for SUT2 is 10. Putting all metrics together, we get the radar chart in Figure 10, showing that SUT2 gives better results on all metrics.

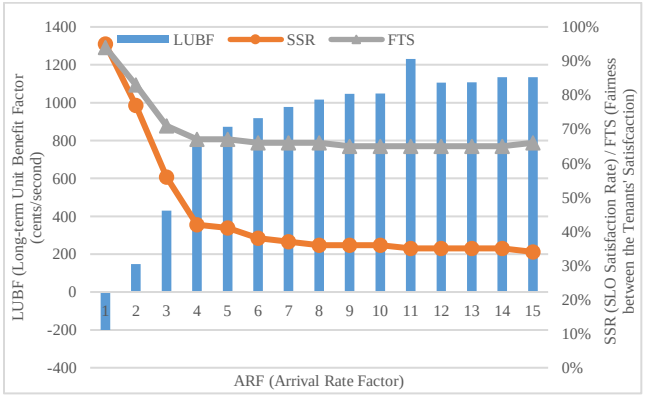


Figure 8: Global trend of the main metrics for SUT1 (RCB)

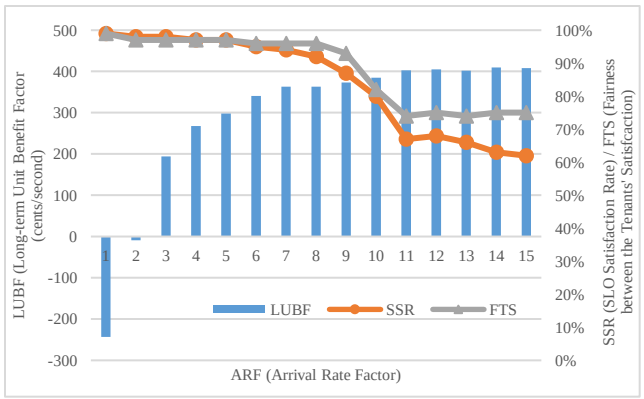


Figure 9: Global trend of the main metrics for SUT2 (RCB)

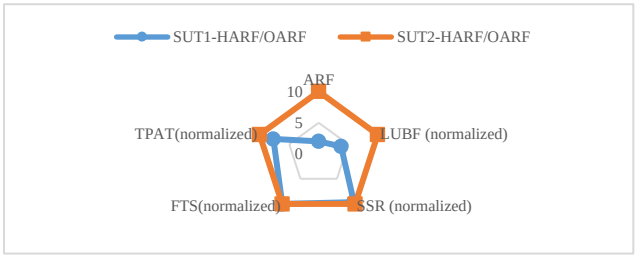


Figure 10: Comparison of SUT1 and SUT2 (RCB)

Independent of system comparing, an interesting observation is that, with the same SUT, the total amount of the tenants' payment increases sometimes while the SLO satisfaction rate decreases, as shown in Figure 11 for SUT2. This is somehow illogical: when the provider receives too many concurrent queries, the execution time of each query may be longer than expected due to interference from other queries; instead of receiving recompense from the provider, the affected tenants need to pay a higher bill. It represents a kind of unfairness in the tenants' point of view. Therefore, we argue that this pricing model is inappropriate for billing database queries, where the resource consumption does not only depend on the problem complexity (e.g., the query type and the database size, etc.), but also depends on the quality of the solution (e.g., query execution plan selection and performance isolation, etc.) which is under the provider's responsibility.

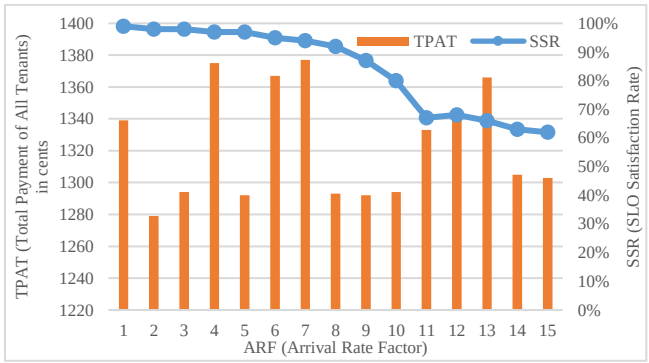


Figure 11: TPAT vs. SSR for SUT2 (RCB)

4.4. Results with the QLSA Pricing Model

With the QLSA pricing model, the price of each query is independent of the resource consumption during the real execution. It is calculated based on a baseline resource consumption, as well as the discrepancy between the real QCT and the expected QCT. The aim is that the tenants do not pay more if it is the provider who fails to execute a query in an optimal way and they could even receive a refund when the performance SLO is not met. Figure 12 shows this consistency for SUT2 by using the QLSA pricing model.

In Figure 13, we can see another change about the LUBF evolution trend (compared to Figure 9): when the ARF grows, the LUBF increases at the beginning because of the system's higher resource utilization ratio, but it decreases later due to penalties caused by SLO violations. In this case, the provider's benefit is more directly linked to the tenants' SLO satisfaction.

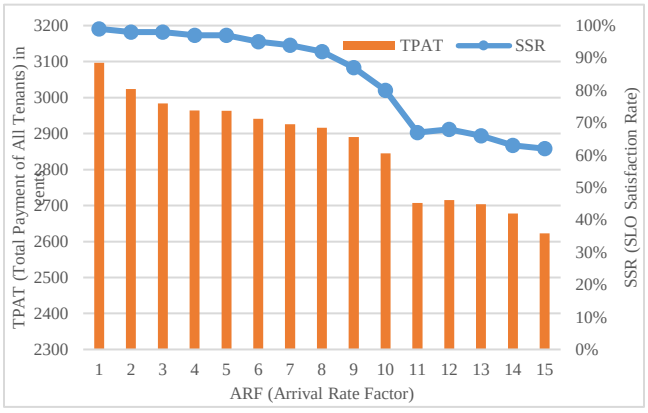


Figure 12: TPAT vs. SSR for SUT2 (QLSA)

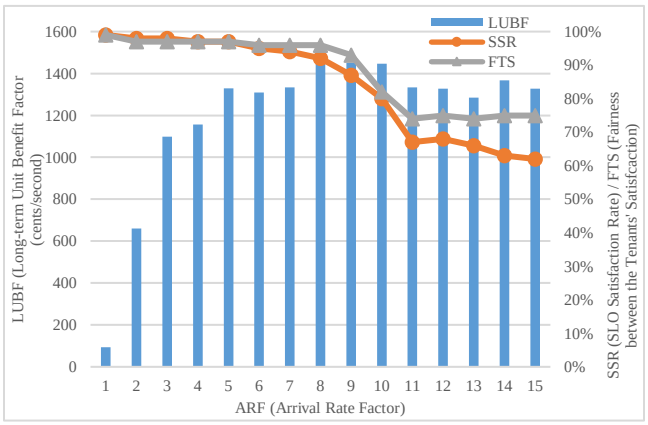


Figure 13: Global trend of the main metrics for SUT2 (QLSA)

Finally, we draw the radar chart for comparing SUTs under the QLSA pricing model in Figure 14. This time, the HARF and OARF have different values for the SUT2. In fact, SUT2 can sustain an ARF of 10 without disappointing too much the tenants (SSR > 70%), but the optimal ARF is 9, where it gives the best LUBF as well as a much better SSR (87%) but with a slightly higher bill for the tenants (2%).

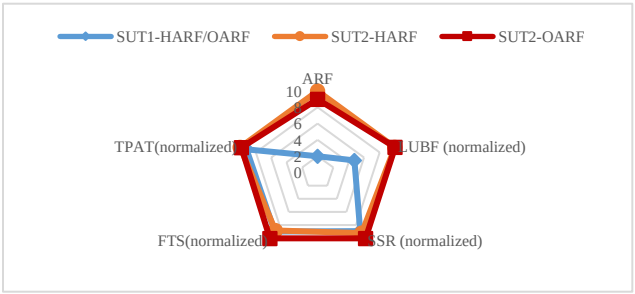


Figure 14: Comparison of SUTs (QLSA)

4.5. Lessons Learned

Below, we summarize some lessons that we have learned through the design of the benchmark and the experiments.

Lesson 1. As a multi-tenant parallel DBMS user, facing the inherent conflict between its benefit and the tenants' performance SLOs, the provider should choose carefully its pricing model in order to allow better trade-offs and fair negotiations (ref. Figures 9 and 13).

Lesson 2. When the provider chooses a multi-tenant parallel DBMS among several candidates, it should be confident that, technically, it is possible that a system (or a version of a system, or even a special way to use a system) could give better results than another one on both tenant-oriented and provider-oriented metrics (ref. Figures 10 and 14).

Lesson 3. Based on the preliminary system comparisons, we believe that, a large spectrum of work could be done on the internal implementations of multi-tenant parallel DBMSs (e.g., admission control, workload management, query optimization and lower level resource allocation) in order to improve both the provider's benefit and the tenants' performance SLOs.

5 CONCLUSION

In this paper, we presented MTD-DS benchmark for multi-tenant parallel DBMSs by extending TPC-DS, in order to include multi-tenancy considerations. Concretely, we have added a multi-tenant query workload generator, a performance SLO (Service Level Objective) generator, some configurable DBaaS pricing models, and new metrics to measure the potential capability of a multi-tenant parallel DBMS in obtaining the best trade-off between the provider's benefit and the tenants' satisfaction.

We conducted experiments with a small but real system to run the proposed benchmark on two variations of the system, in order to demonstrate the feasibility of the benchmark and the relevance of the proposed metrics. Two pricing models were instantiated and examined along with the systems under test. Interesting observations have been made, which allow us a better understanding of what a provider and a multi-tenant parallel DBMS designer could do to offer a better DBaaS product.

ACKNOWLEDGMENTS

This work has been funded by both the MCD project of the LabEx CIMI and the project number 44086TD of the Amadeus program 2020 (French-Austrian Hubert Curien Partnership – PHC). The Amadeus program 2020 is supported by the French Ministries of Foreign and European Affairs and of Higher Education and Research, as well as the International Cooperation & Mobility (ICM) of the Austrian Agency for International Cooperation in Education and Research (OeAD-GmbH).

REFERENCES

[1] Victor Aluko, Sherif Sakr: Big SQL systems: an experimental evaluation. Clust. Comput. 22(4): 1347-1377 (2019).

[2] Amazon Redshift. https://aws.amazon.com/redshift/.

[3] Cloudera Data Warehouse. https://www.cloudera.com/products/data-warehouse.html.

[4] Avrilia Floratou, Umar Farooq Minhas, Fatma Özcan: SQL-on-Hadoop: Full Circle Back to Shared-Nothing Database Architectures. Proc. VLDB Endow. 7(12): 1295-1306 (2014).

[5] Ahmad Ghazal, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte, Hans-Arno Jacobsen: BigBench: towards an industry standard benchmark for big data analytics. SIGMOD Conference 2013: 1197-1208.

[6] Henry Cook, Merv Adrian, Rick Greenwald, Xingyu Gu. 2022 Gartner Magic Quadrant for Cloud Database Management Systems. December 13, 2022.

[7] Mira El Danaoui, Shaoyi Yin, Abdelkader Hameurlain, Franck Morvan. A Cost-Effective Query Optimizer for Multi-tenant Parallel DBMSs. ADBIS 2023: 25-34.

[8] Google BigQuery. https://cloud.google.com/bigquery/.

[9] Randolph W. Hall. Queueing methods: for services and manufacturing. Englewood Cliffs (N.J.): Prentice Hall; 1991.

[10] William H. Inmon. EIS and the data warehouse: A simple approach to building an effective foundation for EIS, Database Programming and Design, 5(11):70-73, 1992.

[11] Shrainik Jain, Dominik Moritz, Daniel Halperin, Bill Howe, Ed Lazowska: SQLShare: Results from a Multi-Year SQL-as-a-Service Experiment. SIGMOD Conference 2016: 281-293.

[12] W. Lang, S. Shankar, J. M. Patel, and A. Kalhan. Towards multi-tenant performance SLOs. In ICDE, pages 702–713, 2012.

[13] Microsoft Azure SQL. https://azure.microsoft.com/en-us/products/azure-sql/.

[14] Vivek R. Narasayya, Surajit Chaudhuri. Multi-Tenant Cloud Data Services: State-of-the-Art, Challenges and Opportunities. SIGMOD Conference 2022: 2465-2473.

[15] Jennifer Ortiz, Victor Teixeira de Almeida, Magdalena Balazinska: Changing the Face of Database Cloud Services with Personalized Service Level Agreements. CIDR 2015.

[16] Raghunath Othayoth Nambiar, Meikel Poess: The Making of TPC-DS. VLDB 2006: 1049-1058.

[17] Meikel Pöss, Raghunath Othayoth Nambiar, David Walrath. Why You Should Run TPC-DS: A Workload Analysis. VLDB 2007: 1138-1149.

[18] Meikel Poess, Tilmann Rabl, Hans-Arno Jacobsen: Analysis of TPC-DS: the first standard benchmark for SQL-based big data systems. SoCC 2017: 573-585.

[19] Nicolás Poggi, Víctor Cuevas-Vicenttín, Josep Lluis Berral, Thomas Fenech, Gonzalo Gómez, Davide Brini, Alejandro Montero, David Carrera, Umar Farooq Minhas, José A. Blakeley, Donald Kossmann, Raghu Ramakrishnan, Clemens A. Szyperski: Benchmarking Elastic Cloud Big Data Services Under SLA Constraints. TPCTC 2019: 1-18.

[20] Oracle Cloud. https://www.oracle.com/cloud/.

[21] Alexander van Renen, Viktor Leis. Cloud Analytics Benchmark. Proc. VLDB Endow. 16(6): 1413-1425 (2023).

[22] ScaleGrid. https://scalegrid.io/

[23] Snowflake. https://www.snowflake.com.

[24] Johann Schleier-Smith, Vikram Sreekanti, Anurag Khandelwal, João Carreira, Neeraja Jayant Yadwadkar, Raluca Ada Popa, Joseph E. Gonzalez, Ion Stoica, David A. Patterson. What serverless computing is and should become: the next phase of cloud computing. Commun. ACM 64(5): 76-84 (2021).

[25] TPC Pricing specification. https://www.tpc.org/tpc_documents_current_versions/pdf/pricing_v2.8.0.pdf.

[26] Adrian Vogelsgesang, Michael Haubenschild, Jan Finis, Alfons Kemper, Viktor Leis, Tobias Mühlbauer, Thomas Neumann, Manuel Then: Get Real: How Benchmarks Fail to Represent the Real World. DBTest@SIGMOD 2018: 1:1-1:6.

[27] Shaoyi Yin, Abdelkader Hameurlain, Franck Morvan: SLA Definition for Multi-Tenant DBMS and its Impact on Query Optimization. IEEE Trans. Knowl. Data Eng. 30(11): 2213-2226 (2018).