

TPCC Benchmark Report



Alibaba Cloud

**TPC Benchmark™ C
Full Disclosure Report**

Alibaba Cloud PolarDB Limitless (with 2,340 PolarDB Data Nodes)

Using

Alibaba Cloud PolarDB for MySQL 8.0.2 Enterprise Edition

First Edition
Jan 19th 2025

First Edition – Jan 19th, 2025

Copyright ©2025 Alibaba Cloud Computing (Hangzhou), Ltd. And/or its affiliates. All rights reserved.

Alibaba Cloud Computing (Hangzhou) Co., Ltd., the Sponsor of this benchmark test, asserts that the pricing information in this document is accurate as of its publication date but is subject to change without prior notice. The Sponsor is not responsible for any errors that may appear in this document. While the pricing information is believed to reflect current prices as of the publication date, the Sponsor makes no guarantees or warranties regarding its accuracy.

The performance information provided in this document is intended for guidance purposes only. Benchmark results are influenced by numerous factors, including workload, hardware, operating environments, specific application requirements, and system design and implementation. Consequently, relative system performance may vary significantly due to these and other variables. The Sponsor makes no guarantee or representation that users will achieve the same or comparable performance metrics, such as transactions per minute (tpmC®) or normalized price/performance (\$/tpmC®). No express or implied warranty regarding system performance is provided.

PolarDB-MySQL is a registered trademark of Alibaba Cloud and/or its affiliates.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. TPC Benchmark, TPC-C, and tpmC are trademarks of the Transaction Processing Performance Council. All other products mentioned herein are trademarks or registered trademarks of their respective owners

©Copyright 2024 Alibaba Cloud Computing (Hangzhou) Co., Ltd.

This product and its accompanying documentation are protected by copyright and distributed under licenses that limit their use, reproduction, distribution, and de-compilation. No portion of this product or its documentation may be reproduced in any form or by any means without prior written permission from Alibaba Cloud Computing (Hangzhou) Co., Ltd. and its licensors, where applicable.

THIS PUBLICATION IS PROVIDED WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.


THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. ALIBABA CLOUD COMPUTING (HANGZHOU) CO., LTD. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT.

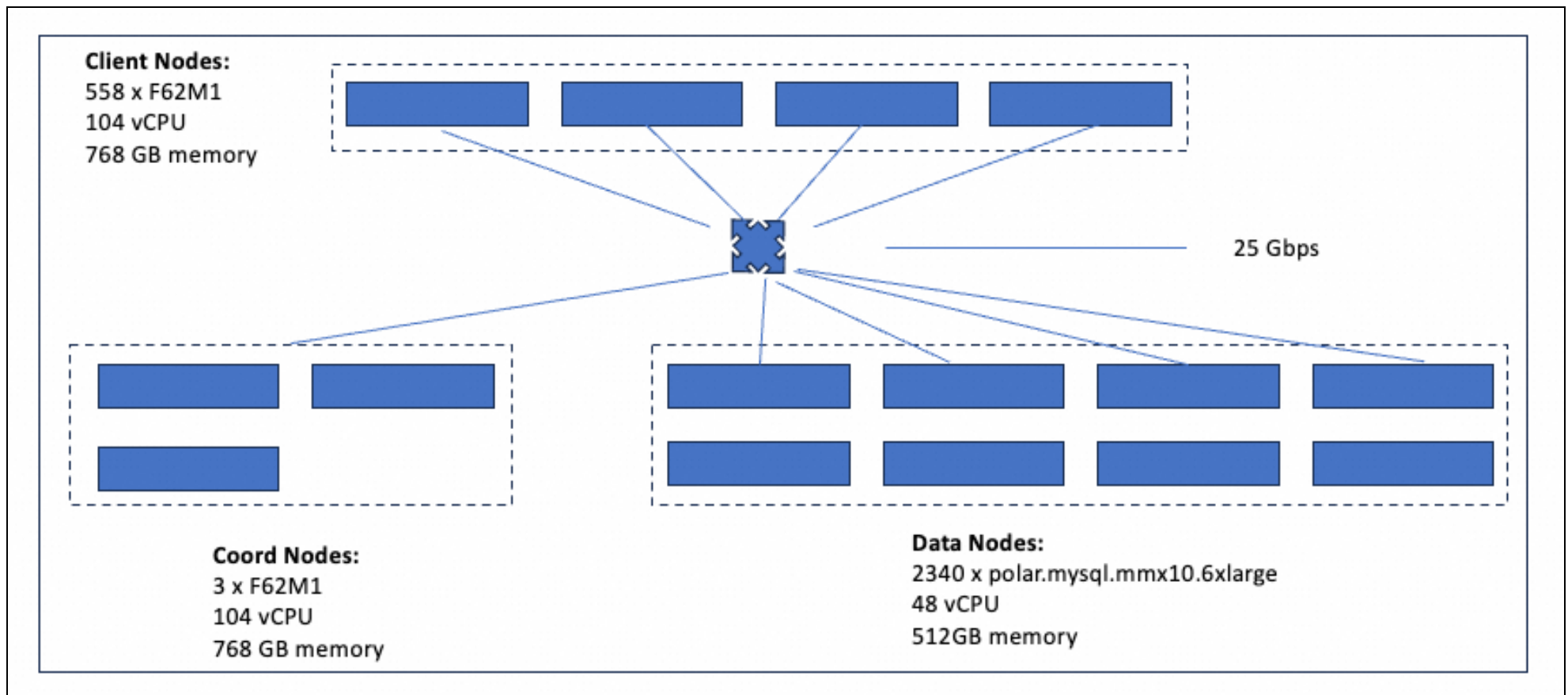
Abstract

This report details the methodology and results of the TPC Benchmark™ C test conducted on the specified environment, as measured by Alibaba Cloud Computing (Hangzhou) Co., Ltd. The benchmark configuration, test environment, methodology used for producing and validating the results, and the pricing model for calculating price/performance were audited by Doug Johnson of InfoSizing to ensure compliance with the applicable TPC specifications.


System	Processors	Database Environment	Operating System
Alibaba Cloud PolarDB Limitless (with 2340 PolarDB Data Nodes)	INTEL(R) XEON(R) PLATINUM 8575C, 3.199GHz, 48 vCPU	Alibaba Cloud PolarDB for MySQL (PolarDB-M) 8.0.2 Enterprise Edition	Alibaba Group Enterprise Linux Server 7.2 (Paladin)

Total System Cost	TPC-C® Throughput	Price / Performance	Availability Date
Three year cost including hardware, software, and maintenance	Maximum Qualified Throughput expressed as transaction per minute - C (tpmC)	Total System Cost / tpmC	Date for which all components, hardware and software are available for purchase
1,626,643,837.00 CNY	2,055,076,649 tpmC	0.80 CNY/tpmC	Jan 19th 2025

	Alibaba Cloud PolarDB Limitless (with 2340 PolarDB for MySQL Data Nodes)		TPC-C 5.11.0 TPC-Pricing 2.9.0	
			Report Date Jan 19th 2025	
Total System Cost	TPC-C Throughput	Price/Performance	Availability Date	
1,626,643,837.00 CNY	2,055,076,649 tpmC	0.80 CNY/tpmC	Jan 19th 2025	
Database Server Cores	Database Manager	Operating System	Other Software	Number of Users
INTEL(R) XEON(R) PLATINUM 8575C, (3.2GHz) 112,320 vCPU	Alibaba Cloud PolarDB for MySQL (PolarDB-M) 8.0.2 Enterprise Edition	Alibaba Group Enterprise Linux Server 7.2 (Paladin)	Nginx 1.25.3.1	1,632,150,000



System Component	2340 Data Nodes		3 Coord Nodes		558 Client Nodes		Total
vCPU	48	INTEL(R) XEON(R) PLATINUM 8575C @ 3.2GHz	104	Intel(R) Xeon(R) Platinum 8269CY @ 2.50GHz	104	Intel(R) Xeon(R) Platinum 8269CY @ 2.50GHz	170,664
Memory	1	512GB	1	768 GB	1	768 GB	1590.75 TB
OS Disk	1	240 GB	1	240 GB	1	240 GB	679.92 TB
Data Disk	1	31.5TB	0	N/A	0	N/A	73,710.00 TB
Total Disk Storage		74220.73 TB		0.66 TB		121.84 TB	74,389.92 TB

	Alibaba Cloud PolarDB Limitless (with 2340 PolarDB for MySQL Data Nodes)					TPC-C 5.11.0 TPC-Pricing 2.9.0
						Report Date Jan 19th 2025
Description	Part Number	Src	Discount	Unit Price	Qty	Ext.Price (CNY)
Alibaba Cloud PolarDB Limitless						
polar.mysql.mmx10.6xlarge (3-year-price)		1		13,200	2,340	1,111,968,000.00
- INTEL(R) XEON(R) PLATINUM 8575C CPU @3.2GHz (48 vCPU)	included	1		0.00	2,340	
- Memory (512G)	included	1		0.00	2,340	
- Alibaba Group Enterprise Linux Server 7.2 (Paladin)	included	1		0.00	2,340	
- Alibaba Cloud PolarDB for MySQL 8.0.2 Enterprise Edition	included	1		0.00	2,340	
- Alibaba Cloud PolarDB Online Support Service 7x24 for 3 years	included	1		0.00	2,340	
					Sub-Total	1,111,968,000.00
PSL3-31.5T (3-year-price)		1		25,232	2,340	2,125,543,680
PSL3 (31.5TB)	included	1		0.00	2,340	
					sub-Total	2,125,543,680.00
Other Services						
OpenResty Professional Edition Support Service for 3 years 7x24x4hrs		2		14,000.00	560	7,840,000.00
- Nginx support included	Included					
					Sub-Total	7,840,000.00
Other Components						
MacBook Pro Laptop (includes 2 sapres)	MCX14LL/A	3		15,999.00	3	47,997.00
					Sub-Total	47,997.00
Discount						
DB-48C512G (3-year-price)		1	50%	-6,600	2340	- 555,984,000
PSL3-31.5T		1	50%	-12,904	2340	- 1,062,771,840
					Sub-Total	-1,618,755,840.00
Src						
					3-Y Cost of Ownership (CNY)	1,626,643,837.00
1. Alibaba Cloud 2. OpenResty Inc 3. www.apple.com.cn					tpmC	2,055,076,649
Audited by Doug Johnson of InfoSizing					CNY/tpmC	0.80

The prices referenced in TPC Benchmarks represent the actual costs a customer would pay for a one-time purchase of the specified components. Individually negotiated discounts and special pricing based on assumptions about past or future purchases are not allowed. All discounts adhere to standard pricing policies for the listed components. For complete details, refer to the pricing section of the TPC benchmark specifications. If you discover that the stated prices are unavailable under these terms, please notify the TPC at pricing@tpc.org. Thank you.


	Alibaba Cloud PolarDB Limitless (with 2340 PolarDB for MySQL Data Nodes)	TPC-C 5.11.0 TPC-Pricing 2.8.0		
		Report Date Jan 19th 2025		
Numerical Quantities:				
MQTH (computed Maximum Qualified Throughput)	2,055,076,649 tpmC			
Ramp-up Time	20 min			
Measurement Interval (MI)	120 min			
Checkpoints in MI per Node (Min/Max/Avg)	125,648/538,640/291275			
Checkpoint Interval (all nodes) Min/Max/Avg	1ms/74696ms/181ms			
Number of transactions all types completed in MI	549,201,278,498			
Response Time (ms)				
	Min Latency	Avg Latency	P90 Latency	Max Latency
New-order	101.030	138.450	163.300	59,999.900
Payment	100.657	137.156	168.100	59,999.900
Order-status	100.800	166.628	190.200	59,999.900
Stock-level	101.185	151.218	175.500	59,999.900
Delivery	100.090	103.037	101.500	11,804.000
Menu	100.071	103.272	101.400	11,813.200
Delivery_deferred	2.408	33.936	61.000	59,999.900
Transaction Mix				
	Number	Percent		
New-order	246,609,197,973	44.903%		
Payment	236,407,712,464	43.046%		
Order-status	22,050,605,728	4.015%		
Stock-level	22,065,766,770	4.018%		
Delivery	22,067,995,563	4.018%		
Keying Times (sec)				
	Min	Avg	Max	
New-order	18.001	18.002	19.893	
Payment	3.001	3.002	4.883	
Order-status	2.001	2.002	3.819	
Stock-level	2.001	2.002	3.816	
Delivery	2.001	2.002	3.827	
Thinking Times (sec)				
	Min	Avg	Max	
New-order	0.000	12.079	122.645	
Payment	0.000	12.079	122.497	
Order-status	0.000	10.086	102.293	
Stock-level	0.000	5.096	52.408	
Delivery	0.000	5.097	52.285	

Table of Contents

TPCC Benchmark Report	1
Preface	9
Introduction	9
0 General Items	9
0.1 Application Code and Definition Statement	9
0.2 Benchmark Sponsor	9
0.3 Parameter Settings	9
0.4 Configuration Diagrams	9
1 Clause 1: Logical Database Design Related Items	11
1.1 Table Definitions	11
1.2 Physical Organization of Database	11
1.3 Insert and Delete Operations	11
1.4 Horizontal or Vertical Partitioning	11
1.5 Replication or Duplication	11
2 Clause 2: Transaction and Terminal Profiles	12
2.1 Random Number Generation	12
2.2 Input/Output Screens	12
2.3 Priced Terminal Feature	12
2.4 Presentation Managers	12
2.5 Transaction Statistics	12
2.6 Queuing Mechanism	13
3 Clause 3: Transaction and System Properties Related Items	13
3.1 Transaction System Properties (ACID)	13
3.2 Atomicity	13
3.2.1 Completed Transaction	13
3.2.2 Aborted Transaction	13
3.3 Consistency	14
3.4 Isolation Tests	14
3.4.1 Isolation Test 1	14
3.4.2 Isolation Test 2	14
3.4.3 Isolation Test 3	15
3.4.4 Isolation Test 4	15
3.4.5 Isolation Test 5	15
3.4.6 Isolation Test 6	15
3.4.7 Isolation Test 7	16
3.4.8 Isolation Test 8	16
3.4.9 Isolation Test 9	16
3.5 Durability	16
3.5.1 Instantaneous Interruption, Memory Failure, Network Failure, Disk Failure, Loss of Log, Loss of Database Tables	16
3.5.2 Power Failure, Full cluster failure	17
4 Clause 4: Scaling and Database Population	17
4.1 Cardinality of Tables	17
4.2 Distribution of tables and logs	17
4.3 Data model and database interface	18
4.4 The mapping of database partitions/replications	18
4.5 60 Day Space Computation	18
5 Clause 5: Performance Metrics and Response Time Related Items	19
5.1 Measured tpmC	19
5.2 Response Times	19
5.3 Keying and Think Times	19
5.4 Response Time Frequency Distribution Curves	20
5.5 Think Time Frequency Distribution	22
5.6 Response Times versus Throughput	22
5.7 Throughput versus Elapsed Time	23
5.8 Steady State Determination	23
5.9 Work Performance During Steady State	24
5.10 Measurement Interval	24

5.11 Transaction Mix Regulation	24
5.12 Transaction Mix	24
5.13 Percentage of New-Order Transactions	24
5.14 Number of Order-lines per New-Order	24
5.15 Percentage of Remote Order-lines per New-Order	25
5.16 Percentage of Remote Payments.....	25
5.17 Percentage of access customer by C_LAST for Payment and Order-Status.....	25
5.18 Percentage of Skipped Delivery Transactions.....	25
5.19 Checkpoints	25
6 Clause 6: SUT, Driver and Communications Related Items	26
6.1 RTE Description.....	26
6.2 Number of Terminal Connection Lost	26
6.3 Emulated Components.....	26
6.5 Network Configuration	26
6.6 Operator Intervention.....	26
7 Clause 7: Pricing Related Items	27
7.1 Hardware and software Price.....	27
7.2 Total 3-Year Cost	27
7.3 Availability Date	27
7.4 Hardware and Software Support.....	27
7.5 Statement of measured tpmC and Price/Performance	27
7.6 Country Specific Pricing	27
7.7 Orderability Date	27
8 Clause 8: Auditor Attestation	27
8.1 Auditor Information.....	27
8.2 Attestation Letter.....	28
Appendix A: Source Code File List.....	30
Appendix B: Database design	31
B.1 Table creation	32
B.2 Procedure creation	33
Appendix C: Configuration Options	42
C.1 F62M1 OS configuration	42
C.2 Nginx configuration	42
C.3 Data Node database configuration.....	43
Appendix D: Price References.....	45
D.1 Alibaba Cloud PolarDB Limitless with 2340 polar.mysql.mmx10.6xlarge and 31.5TB storage each.....	45
D.2 14 英寸 MacBook Pro.....	45
D.3 OpenResty Support Price.....	46

Preface

The TPC Benchmark™ C Standard Specification was developed by the Transaction Processing Performance Council (TPC). It was initially released on August 13, 1992, and later updated to Revision 5.11 in February 2010. This report serves as the full disclosure document for benchmark testing conducted on Alibaba Cloud PolarDB for MySQL 8.0.2 Enterprise Edition, running on Alibaba Cloud Service with 2,340 PolarDB data nodes, in compliance with the TPC Benchmark C Standard Specification, Revision 5.11.

TPC Benchmark™ C Standard

The main section of this document addresses each item outlined in Clause 8 of the TPC Benchmark™ C Standard, detailing how the corresponding specifications are met.

- Appendix A contains the application source code file list.
- Appendix B contains the SQL queries used to create tables and procedures.
- Appendix C contains the configuration information for all the software and operating systems.
- Appendix D contains the Price References including those from the third party

Introduction

The TPC Benchmark™ C Standard Specification mandates that test sponsors publish and publicly share a full disclosure report for the results to be deemed compliant with the Standard.

This report is designed to fulfill the Standard's requirement for full disclosure. It provides documentation of the benchmark tests' compliance with the TPC Benchmark™ C requirements for Alibaba Cloud PolarDB for MySQL Limitless, featuring 2,340 database nodes running PolarDB for MySQL (PolarDB-M) 8.0.2 Enterprise Edition.

In the Standard Specification, the main headings in Clause 8 are aligned with other clauses. This report follows the same sequence of headings to correspond with the titles or subjects referenced in Clause 8.

Each section of this report begins with the relevant item from Clause 8 of the Standard Specification, presented in italic type. The following plain text provides an explanation of how the tests comply with the TPC-C Benchmark.

0 General Items

0.1 Application Code and Definition Statement

The application program (as defined in clause 2.1.7) must be disclosed. This includes, but is not limited to, the code implementing the five transactions and the terminal input output functions.

Appendix A includes a list of application source code files for the five TPC-C transactions, covering the input and output from the terminal, database interactions and communication, and the implementation of deferred delivery. All these files are available in the supporting file package.

0.2 Benchmark Sponsor

A statement identifying the benchmark sponsor(s) and other participating companies must be provided.

The benchmark test was sponsored by Alibaba Cloud Computing (Hangzhou), with the implementation developed and engineered by the PolarDB team in the Database R&D Department of Alibaba Inc.

0.3 Parameter Settings

Settings must be provided for all customer-tunable parameters and options which have been changed from the defaults found in actual products, including by not limited to:

- Database tuning options
- Recover/commit options
- Consistency/locking options
- Operating system and application configuration parameters

Appendix C includes the tuned parameters for the operating systems, RTEs, Nginx servers, and database systems.

0.4 Configuration Diagrams

Diagrams of both measured and priced configurations must be provided, accompanied by a description of the differences.

RTE/Client/COORD nodes are all V6 machines. The hardware configuration is listed below:

Table 1: Machine type 1 - F62M1

Physical Machines	For RTE/Client/COORD nodes
CPU	Intel(R) Xeon(R) Platinum 8269CY CPU @ 2.50GHz
Sockets/cores/threads	2/52/104

Memory	768 GB
OS disk	240 GB

Table 2: Machine type 2 – V83S1

Physical Machines	For Data nodes
CPU	INTEL(R) XEON(R) PLATINUM 8575C @ 3.20GHz
Sockets/cores/threads	2/96/192
Memory	2,048 GB (32x 64 GB)
OS Disk	1x 240 GB SSD
Backup Disk	2x 800 GB NVMe
Data Disks	14x 9.6 TB NVMe

Each physical server hosts 4 DB node instances (2 primary nodes, and 2 Standby nodes) where each is configured as follows (i.e., polar.mysql.mmx10.6xlarge).

DB Node Instance	
vCPU	48
Memory	512 GB
Data Storage	31.5 TB

There are 840 servers of Machine Type 1 for RTE/Web/Coordinator nodes and 1,170 servers of Machine Type 2 for the 2,340 DB Node instances.

There are four types of roles:

1. The Remote Terminal Emulator (RTE), which simulates clients, and the System Under Test (SUT), which consists of three components.
2. The Client nodes for web services that handle client requests and communicate with the databases.
3. The COORD nodes, responsible for managing meta information such as routing tables.
4. The Data nodes, where the database is deployed for data storage and processing. There are two types of Data nodes: the primary (RW) node and the replica (Standby) node. The Standby node is not accessible to clients and is used only for recovery if the RW node fails. Therefore, we will refer to the Data node as the RW node in the remainder of the document.

In total, there are 279 instances for RTEs, 558 instances for Client nodes, 3 instances for COORD nodes, and 2,340 DB nodes (i.e., RW nodes).

Figure 1 illustrates the system architecture overview. At the highest level, there are 279 RTE instances representing the clients. In the middle level, 558 client instances handle the requests, convert some tasks into SQL queries, and forward them to the databases. At the lowest level, 3 COORD nodes store and update metadata, while 2,340 Data nodes manage and process the transactional data.

All tested machines are located within a network zone organized by a cluster of switches.

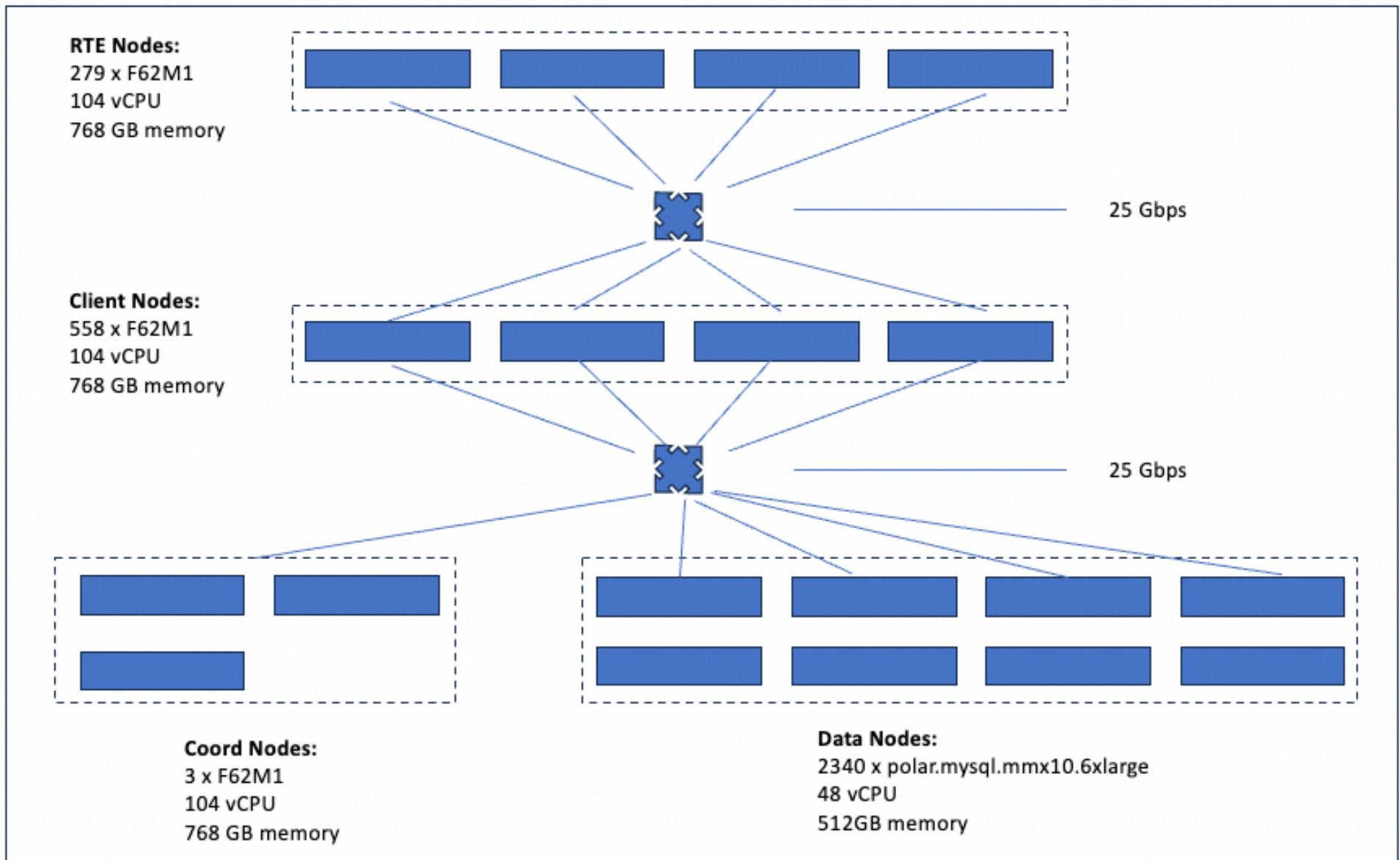


Figure 1, Measured Configuration

The Priced Configuration is shown in the Executive Summary.

1 Clause 1: Logical Database Design Related Items

1.1 Table Definitions

Listing must be provided for all table definition statements and all other statements used to set up the database.

Appendix B includes the scripts used for creating tables and procedures.

1.2 Physical Organization of Database

The physical organization of tables and indices within the database must be disclosed.

Figure 1 and Section 0.4 provide an overview of the SUT environment and system architecture. As a cloud database, PolarDB separates computing from storage and charges for storage based on actual usage (the total local disk space exceeds actual usage to accommodate space expansion).

All database objects, including tables, indexes, and logs, are stored on these local disks. Appendix B includes the scripts used to create tables and indexes. When creating tables, the partition key is specified to ensure the tables are partitioned.

1.3 Insert and Delete Operations

It must be ascertained that insert and/or delete operations to any of the tables can occur concurrently with the TPC-C transaction mix. Furthermore, any restrictions in the SUT database implementation that precludes inserts beyond the limits defined in Clause 1.4.11 must be disclosed. This includes the maximum number of rows that can be inserted and the minimum key value for these new rows.

The auditor verified that all insert and delete functions were fully operational throughout the entire benchmark.

1.4 Horizontal or Vertical Partitioning

While there are a few restrictions placed upon horizontal or vertical partitioning of tables and rows in the TPC-C benchmark, any such partitioning must be disclosed.

The ITEM table is a global table that is fully replicated across all data nodes. All other tables are partitioned horizontally, with the partitions defined during table creation by configuring the partition key parameter in the PolarDB database system.

Table 3. Summary of Partitioning

Table Name	Table Type	Partition Key
bmsql_warehouse	partition table	w_id
bmsql_district	partition table	d_w_id
bmsql_customer	partition table	c_w_id
bmsql_history	partition table	h_c_w_id
bmsql_new_order	partition table	no_w_id
bmsql_oorder	partition table	o_w_id
bmsql_order_line	partition table	ol_w_id
bmsql_stock	partition table	s_w_id
bmsql_item	global table	

Note:

- partition table: The table is split into all sets using the partition key field
- global table: All sets have a copy of the full table data

1.5 Replication or Duplication

Replication of tables, when utilized, must be disclosed. Any additional or duplicated attributes within a table must also be disclosed, along with an explanation of their impact on performance.

The ITEM table is fully replicated across all 2,340 data nodes. All other tables are replicated into two replicas, each hosted on different machines. The primary node, which provides read-write services, is called the RW-node, while the backup node for failover is referred to as the Standby-node. For instance, RW-node1 and RW-node2 are hosted on physical machine 1, with their corresponding Standby-nodes (replicas) hosted on physical machine 2.

RW-nodes and Standby-nodes are distributed in a round-robin manner across physical machines. The replicas of RW-node 2379 and RW-node 2340 will be hosted on physical machine 1.

The RW-node contains persistent data, page cache, continuous checkpoints, and redo/undo logs. Similarly, the Standby-node contains persistent data, page cache, continuous checkpoints, and redo/undo logs. The Standby-node is synchronized with the RW-node through the redo log (i.e., PolarDB physical replication).

2 Clause 2: Transaction and Terminal Profiles

2.1 Random Number Generation

The method of verification for the random number generation must be described.

The default rand package in Go is used to implement a pseudorandom number generator, which is seeded with the current system timestamp.

2.2 Input/Output Screens

The actual layout of the terminal input/output screens must be disclosed.

The auditor verified all screen layouts to ensure they met the specifications' requirements. The source code for generating the screens is available and provided. Appendix A contains the list of application source code files.

2.3 Priced Terminal Feature

The method used to verify that the emulated terminals provide all the features described in Clause 2.2.2.4 must be explained. Although not specifically priced, the type and model of the terminals used for the demonstration in 8.1.3.3 must be disclosed and commercially available (including supporting software and maintenance).

The auditor verified the mechanism of the emulated terminals to ensure that each required feature was properly implemented.

2.4 Presentation Managers

Any usage of presentation managers or intelligent terminals must be explained.

No intelligent terminal is used. The database and SUT generate test results in HTML format, which can be viewed in any web browser. Appendix A includes the source code files for creating the HTML display. In the past, transactions were initiated through a web page.

2.5 Transaction Statistics

The percentage of home and remote order-lines in the New-Order transactions must be disclosed

The percentage of New-Order transactions that were rolled back as a result of an unused item number must be disclosed.

The number of items per orders entered by New-Order transactions must be disclosed

The percentage of home and remote Payment transactions must be disclosed.

The percentage of Payment and Order-Status transactions that used non-primary key (C_LAST) access to the database must be disclosed.

The percentage of Delivery transactions that were skipped as a result of an insufficient number of rows in the NEW- ORDER table must be disclosed.

The percentage of Delivery transactions that were skipped as a result of an insufficient number of rows in the NEW- ORDER table must be disclosed.

The mix (i.e., percentages) of transaction types seen by the SUT must be disclosed.

The table below outlines the percentage distribution of the specified transactions.

Table 4: Statistics for transactions and terminals

New Order	
Percentage of Home order-line	99.048%
Percentage of remote order-line	0.952%
Percentage of Rolled Back Transactions	1.004%

Avg. Number of Items per Transactions	10.00
Payment	
Percentage of Home Transactions	85.000%
Percentage of Remote Transactions	15.000%
Access by C_LAST (Non-primary key)	
Percentage of Payment Transactions	60.000%
Percentage of Order-Status Transactions	60.000%
Delivery	
Skipped Transaction	0.00
Transaction Mix	
New-Order	44.903%
Payment	43.046%
Order-status	4.015%
Delivery	4.018%
Stock-level	4.018%

2.6 Queuing Mechanism

The queuing mechanism used to defer the execution of the Delivery transaction must be disclosed.

The Delivery Queue has been implemented in the SUT. When the SUT receives a Delivery request from the RTE, it is placed into the Delivery Queue and immediately sends a response back. Within the SUT, routines periodically check the Delivery Queue for pending requests, execute the deferred Delivery transactions, and log the results into the result file.

3 Clause 3: Transaction and System Properties Related Items

3.1 Transaction System Properties (ACID)

The results of the ACID tests must be disclosed along with a description of how the ACID requirements were met. This includes disclosing which case was followed for the execution of Isolation Test 7.

To ensure the System Under Test (SUT) aligns with the essential principles of Atomicity, Consistency, Isolation, and Durability (ACID), it's necessary to perform a series of evaluations. This part of the report outlines the specific tests that were executed and demonstrates how PolarDB adheres to the standards established in the official guidelines.

3.2 Atomicity

The system under test must guarantee that the database transactions are atomic. The system will either perform all individual operations on the data or will assure that no partially completed operations leave any effects on the data.

3.2.1 Completed Transaction

Perform the Payment transaction for a randomly selected warehouse, district, and customer (by customer number as specified in Clause 2.5.1.2) and verify that the records in the CUSTOMER, DISTRICT, and WAREHOUSE tables have been changed appropriately

The atomicity test for Alibaba Cloud PolarDB Limitless was conducted twice to evaluate different scenarios. For the initial test simulating a local environment, the payment transaction occurred within a single Data node. This setup ensured that the warehouse, district, and customer tables, chosen randomly, were housed on the same node. In the subsequent test, designed to mimic remote conditions, the transaction was distributed across two Data nodes, requiring that the randomly selected warehouse, district, and customer tables be stored on separate nodes.

To ensure the atomicity of a completed Payment transaction, a series of steps were conducted. Initially, a row was chosen randomly from the warehouse, district, and customer tables in Alibaba Cloud PolarDB Limitless, and the balance was recorded as B1. Subsequently, a payment transaction was initiated using the previously selected warehouse, district, and customer, along with a specified balance amount (b0), following which the transaction was committed. It was confirmed that the balance following the transaction (B2) equaled the original balance minus the specified amount, indicating that $B2 = B1 - b0$.

3.2.2 Aborted Transaction

Perform the Payment transaction for a randomly selected warehouse, district, and customer (by customer number as specified in Clause 2.5.1.2) and substitute a ROLLBACK of the transaction for the COMMIT of the transaction. Verify that the records in the customer, district and warehouse tables have NOT been changed.

The atomicity test was carried out twice using Alibaba Cloud PolarDB Limitless. The first scenario mimicked a local environment where the payment transaction was conducted on a single data node of Alibaba Cloud PolarDB Limitless. In this scenario, the warehouse, district, and customer tables, chosen randomly, were all located on the same node. In the second scenario, designed to simulate a remote setup, the transaction spanned two data nodes, meaning that the randomly chosen warehouse, district, and customer tables were distributed across two separate nodes.

To confirm the atomicity of an aborted Payment transaction, the following steps were taken: a row was randomly picked from the warehouse, district, and customer tables within the Alibaba Cloud PolarDB Limitless, and an initial balance (B1) was recorded. A payment transaction was started using the selected warehouse, district, and customer, with a specified balance amount (b0), and then the transaction was rolled back. It was then verified that the balance following the rollback (B2) matched the initial balance, thereby confirming $B2 = B1$.

3.3 Consistency

Consistency is the property of the application that requires any execution of a database transaction to take the database from one consistent state to another, assuming that the database is initially in a consistent state. Verify that the database is initially consistent by verifying that it meets the consistency conditions defined in Clauses 3.3.2.1 to 3.3.2.4. Describe the steps used to do this in sufficient detail so that the steps are independently repeatable.

The specification mandates a clear demonstration of four specific consistency tests.

1. The cumulative year-to-date balances (D_YTD) across all districts within a given warehouse need to match the warehouse's own year-to-date balance (W_YTD).
2. Each district within a warehouse must ensure that the next available Order ID (D_NEXT_O_ID) minus one equals both the latest Order ID [$\max(O_ID)$] in the Order table and the most recent Order ID [$\max(NO_O_ID)$] in the New-Order table for that specific district and warehouse. This can be expressed as: $D_NEXT_O_ID - 1 = \max(O_ID) = \max(NO_O_ID)$, where $(D_W_ID = O_W_ID = NO_W_ID)$ and $(D_ID = O_D_ID = NO_D_ID)$.
3. Within each district of a warehouse, the difference between the highest and lowest Order ID [$\max(NO_O_ID) - \min(NO_O_ID)$] in the New-Order table, plus one, must correspond to the total row count of the New-Order table for that district. This can be represented as: $\max(NO_O_ID) - \min(NO_O_ID) + 1 =$ the number of rows in the New-Order table for the district.
4. For every district in a warehouse, the total number of order line counts [$\sum(O_OL_CNT)$] in the Order table associated with the district should equal the number of rows in the Order-Line table associated with the district, illustrated by: $\sum(O_OL_CNT) =$ row count in the Order-Line table for the district, with $(O_W_ID = OL_W_ID)$ and $(O_D_ID = OL_D_ID)$.

The above consistency checks were verified through four different functions independently using a Go program, which issued queries to the Alibaba Cloud PolarDB Limitless database. These consistency checks were carried out twice: once following data loading and once after the performance run report. The resulting queries confirmed that the database maintained consistency throughout all four checks.

3.4 Isolation Tests

The Benchmark Standard Specification in Clause 3.4.2 defines nine tests used to demonstrate the required level of transaction isolation is met in the SUT database.

A comprehensive series of nine unique isolation assessments were conducted using the Alibaba Cloud PolarDB Limitless setup. This involved utilizing a configuration consisting of 2,340 database nodes to manage a massive scale of 163,215,000 warehouse records. The evaluations consistently employed the Repeatable Read (RR) isolation level to maintain uniform testing conditions across all tests.

3.4.1 Isolation Test 1

This test demonstrates isolation for read-write conflicts of Order-Status and New-Order transactions when the New-Order transaction is committed.

The test proceeds as follows.

1. An Order-Status transaction, identified as T0, was executed and committed for a randomly chosen customer, C. The committed state of T0 was confirmed.
2. A New-Order transaction T1 for customer C was initiated, executed, and placed in a 20-second pause before committing.
3. 10 seconds after the start of T1, an Order-Status transaction T2 was commenced, executed, and committed successfully before T1 was committed. T2 was not blocked by T1 and reflected the same order results as T0.
4. The New-Order transaction T1 was then committed.
5. An Order-Status transaction T3 was executed for customer C, reflecting the order entered by T1.

3.4.2 Isolation Test 2

This test demonstrates isolation for read-write conflicts of Order-Status and New-Order transactions when the New-Order transaction is rolled back.

The test proceeds as follows.

1. For a randomly chosen customer C, the Order-Status transaction T0 was executed and committed, confirming its committed state.
2. The New-Order transaction, T1, for customer C was initiated, executed, and paused for 20 seconds before rollback.
3. An Order-Status transaction marked as T2 started 10 seconds after the initiation of T1 and was committed before T1's rollback. T2 was not blocked by T1, returning identical order details as T0.
4. T1 was then allowed to ROLLBACK.
5. Another Order-Status transaction, T3, was executed for customer C, and it returned the same order data as T0.

3.4.3 Isolation Test 3

This test demonstrates isolation for write-write conflicts of two New-Order transactions when both transactions are committed.

The test proceeds as follows.

1. Initiating a transaction T0, the D_NEXT_O_ID for a randomly chosen district D was retrieved and committed.
2. A New-Order transaction T1 was initiated for randomly selected customer C in district D and was paused for 20 seconds before committing.
3. A second New-Order transaction, T2, was initiated 10 seconds after T1's initiation for customer C and awaited processing.
4. After the 20-second pause, T1 was committed, returning the order number which was the same as the D_NEXT_O_ID retrieved in T0.
5. Subsequently, T2 was committed, returning an order number incremented by one from T1's returned value.
6. Another transaction, T3, retrieved the D_NEXT_O_ID of district D, which was observed to be one incremented greater than that returned by T2.

3.4.4 Isolation Test 4

This test demonstrates isolation for write-write conflicts of two New-Order transactions when one transaction is rolled back.

The test proceeds as follows.

1. A transaction T0 was initiated to retrieve the D_NEXT_O_ID of a randomly selected District D and committed.
2. A New-Order transaction T1 for randomly selected customer C in district D was initiated, pausing for 20 seconds before a rollback.
3. 10 seconds after T1's initiation, a second New-Order transaction, T2, was initiated and awaited execution for customer C.
4. Following a 20-second wait, T1 was aborted. The order number it returned matched the D_NEXT_O_ID fetched by T0.
5. T2 was committed, returning an order number identical to the D_NEXT_O_ID obtained by T0.
6. Transaction T3 retrieved the district D's D_NEXT_O_ID, revealing a value one greater than that returned by T2.

3.4.5 Isolation Test 5

This test demonstrates isolation for write-write conflicts of Payment and Delivery transactions when Delivery transaction is committed.

The test proceeds as follows.

1. Transaction T0 executed a query to identify customer C who is to be updated by the forthcoming Delivery transaction for a randomly chosen warehouse W and district D, fetching C's balance (C_BALANCE).
2. A Delivery transaction T1 for warehouse W was initiated and paused for 20 seconds prior to committing.
3. Payment transaction T2 began 10 seconds after T1's start for customer C, entering a wait state.
4. T1 committed post a 20-second wait, followed by the commitment of T2.
5. Subsequently, transaction T3 retrieved C's C_BALANCE once again. The balance matched the outcomes from both T1 and T2, expressed as $C_BALANCE\ in\ T3 = C_BALANCE\ in\ T0 + \text{delivery amount from T1} - \text{payment amount from T2}$.

3.4.6 Isolation Test 6

This test demonstrates isolation for write-write conflicts of Payment and Delivery transactions when the Delivery transaction is rolled back.

The test proceeds as follows.

1. A Transaction T0 was initiated to execute a query identifying customer C who is to be updated by a subsequent Delivery transaction for a selected warehouse W and district D, and collected the initial C_BALANCE.
2. Delivery transaction T1 initiated for warehouse W, pausing for 20 seconds before rolling back.
3. Payment transaction T2 was initiated 10 seconds after the start of T1 for customer C, maintaining a wait condition.
4. T1 was aborted following a 20-second delay, while T2 proceeded to commitment.

- Transaction T3 accessed C's C_BALANCE again. This balance reflecting only T2's effects, was captured as C_BALANCE in T3 = C_BALANCE in T0 - payment amount from T2.

3.4.7 Isolation Test 7

This test demonstrates repeatable reads for the New-Order transaction while an interactive transaction updates the prices of some items.

The test proceeds as follows.

- Transaction T1 initiated to fetch the prices (I_PRICE) of two randomly chosen items, X and Y.
- A New-Order transaction T2 involving items X and Y started, pausing for 20 seconds between its initial and repeated price retrievals for these items.
- 10 seconds after T1's initiation, transaction T3 was initiated. T3 increased the prices of X and Y by 10%, and committed. (Case D is used in the follow-up steps)
- T2 obtained prices of X and Y once again, which matched initial values captured in T1.
- Transaction T2 achieved commitment.
- Transaction T4 revisited items X and Y, finding prices aligned with T3's 10% increments.

3.4.8 Isolation Test 8

This test demonstrates isolation for phantom protection between New-Order and Delivery transactions.

The test proceeds as follows.

- Transaction T0 was initiated to modify the District ID (NO_D_ID) across all New-Order entries to an alternate ID D' (e.g., 127) for a selected warehouse W and district D and committed.
- Delivery transaction T1 for warehouse W started, and paused for 20 seconds after retrieving the NEW_ORDER table for warehouse W and district D. No qualified row was found.
- 10 seconds after T1's start, New-Order transaction T2 was initiated and blocked by T1. T1 repeated the read of the NEW_ORDER table and there was still no qualifying row.
- T1 was committed. After T1 was committed, T2 was unblocked and committed.
- Transaction T3 reinstated the District ID values in the NEW_ORDER table altered by T0.

3.4.9 Isolation Test 9

This test demonstrates isolation for phantom protection between New-Order and Order-Status transactions.

The test proceeds as follows.

- Order-Status transaction T1 initiated for a randomly selected customer C.
- T1 paused for 20 seconds after retrieving the customer's recent order.
- A New-Order transaction T2 was initiated 10 seconds after T1 began and committed without encountering T1-induced blockage. (Case B)
- T1 resumed to fetch the most current order for customer C again. The result was the same as the one found in step 2.
- T1 was committed.

3.5 Durability

The tested system must guarantee durability: the ability to preserve the effects of committed transactions and insure database consistency after recovery from any one of the failures listed in Clause 3.5.3.

3.5.1 Instantaneous Interruption, Memory Failure, Network Failure, Disk Failure, Loss of Log, Loss of Database Tables

To showcase the durability of PolarDB in scenarios involving hardware failures, such as memory, disk, or network failures, as well as software failures, including file loss or instance interruptions. The test was executed by the following steps:

- Verified the consistency of the database and confirmed it passed.
- Queried the sum of D_NEXT_O_ID from all rows in the DISTRICT table to determine the current total number of orders, which is recorded as Count1.
- Started the RTE with full user load.

4. The test was ramped up and ran in steady state at a tpmC greater than 90% of the reported tpmC for at least 20 minutes.
5. Logon Alibaba Cloud control manager and destroyed two DB nodes (i.e., force to shutdown a physical machine that hosts two DB nodes).
6. The Alibaba Cloud PolarDB Limitless recovered automatically within several minutes. The Standby Data nodes replaced them automatically.
7. The RTE reported some failure transaction and recover to steady state in several minutes. The test continued in steady state at a tpmC greater than 90% of the reported tpmC for at least 5 minutes.
8. Logon Alibaba Cloud control manager and destroyed a primary COORD node.
9. The Alibaba Cloud PolarDB Limitless recovered automatically within several seconds. The duplication Manager node replaced it automatically.
10. The RTE does not report some errors. The test ran in steady state at a tpmC greater than 90% of the reported tpmC for at least 5 minutes.
11. Logon Alibaba Cloud control manager and destroyed a Client node.
12. The client node did not recover.
13. The RTE reported failure transactions.
14. The RTE recorded all successful and rollbacked New-Order transactions in a success file during testing period.
15. Queried the sum of D_NEXT_O_ID from all rows in the DISTRICT table again as the ending total number of orders, which is recoded as Count2.
16. Verified consistency of the database again and passed it.
17. Verified that all successful New-Order Transactions in the success file are in the database and all rollbacked transactions in the success file are not in the database.
18. Compared the difference between the counts, i.e., Count2-Count1, with the sum of the committed transactions in the success file on RTE, as Scnt. Checked that Count2-Count1 >= Scnt.

3.5.2 Power Failure, Full cluster failure

PolarDB for MySQL uses two replicas for a single data SET, where a data SET consists of a primary (RW) and a replication (Standby). Each transaction log will be written to both replicas to ensure that the transaction data can be fully recovered from the Standby node in case of the failure of the RW node while guaranteeing the reliability and correctness of the SET data. The two-phase commit protocol is adopted for distributed transactions. For pending transactions, the supplementary commit or rollback will be automatically performed to ensure that the complete ACID feature can be meet in distributed scenarios.

This full cluster failure test is justified by documentation. Alibaba Cloud's physical machines all comply with 30 minutes of Uninterruptible Power Supply (UPS) requirement.

4 Clause 4: Scaling and Database Population

4.1 Cardinality of Tables

The cardinality (e.g., the number of rows) of each table, as it existed at the start of the benchmark run (see Clause 4.2), must be disclosed. If the database was over -scaled and inactive rows of the WAREHOUSE table were deleted (see Clause 4.2.2), the cardinality of the WAREHOUSE table as initially configured and the number of rows deleted must be disclosed.

During the test, there are 163,215,000 warehouses in the PolarDB for MySQL databases. The following table presents the initial cardinality of the tables after population and the cardinality just before the measurement run.

Table 5: Table cardinality

Table	Initial Row Count	Row Count prior to Measure Run
Warehouse	163,215,000	163,215,000
District	1,632,150,000	1,632,150,000
Customer	4,896,450,000,000	4,896,450,000,000
History	4,896,450,000,000	4,896,450,000,000
Order	4,896,450,000,000	4,896,450,000,000
New Order	1,468,935,000,000	1,468,935,000,000
Order Line	48,964,501,163,163	48,964,501,163,163
Stock	16,321,500,000,000	16,321,500,000,000
Item	100,000	100,000

4.2 Distribution of tables and logs

The distribution of tables and logs across all media must be explicitly depicted for the tested and priced systems.

Data is automatically distributed evenly across different data nodes in the PolarDB for MySQL cluster using a partition key. All data is stored on these data nodes. When a table is created with a partition key, the PolarDB cluster automatically splits large tables horizontally into multiple datasets by setting the partition key. Both data distribution and access are based on the hashing of the partition key. Each dataset within the PolarDB cluster consists of one read-write (RW) node and one standby node. The RW-node is synchronized with the Standby-node by replicating the REDO log (i.e., PolarDB physical replication). Both the RW-node and the Standby-node contain data and log files. If any RW-node fails, the Standby-node will take over as the new RW-node. All tables, except the ITEM table, are horizontally partitioned using the partition key. The ITEM table does not use a partition key and is stored as a global table across all datasets in the cluster.

4.3 Data model and database interface

A statement must be provided that describes:

1. The data model implemented by the DBMS used (e.g., relational, network, hierarchical).
2. The database interface (e.g., embedded, call level) and access language (e.g., SQL, DL/1, COBOL read/write) used to implement the TPC-C transactions. If more than one interface/access language is used to implement TPC-C, each interface/access language must be described and a list of which interface/access language is used with which transaction type must be disclosed

PolarDB Limitless is a distributed relational database management system. The RTE module and the SUT's web request processing are developed using the Go programming language, while the database components are written in C/C++. The web server interacts with the database using standard SQL, including DML and stored procedures.

The Client node receives all incoming requests and forwards them to the back-end database layer using the shard key. Upon receiving a request, a Data node executes the corresponding SQL statements and sends the results back to the Client node. The Client node then consolidates the responses from the database layer and relays them to the RTE via the Web Server. The application code files are listed in Appendix A.

4.4 The mapping of database partitions/replications

The mapping of database partitions/replications must be explicitly described.

All tables are horizontally partitioned, with each partition has two replicas distributed across different nodes. The exception is the ITEM table, which is fully replicated on every data node. Each Data node has a set of 2 replicas, and each set is responsible for 25 partitions. Hence there are total 58,500 unique partitions (i.e., 2340 (i.e., number of Data nodes) * 25). Specifically, the rows in the horizontally partitioned tables are hashed using a partition key field (e.g., w_id), the corresponding partition ID (i.e., partition_id) is obtained by the modulo of 58,500 (e.g., w_id%58500), and then the corresponding Data node ID (i.e., master_id) is obtained by the modulo of number of Data Nodes (e.g., partition_id%2340). The IP address/port number of the specific database is obtained by querying information from the routing table on COORD node with Data node ID (i.e., master_id).

Warehouse		District		Customer		History		Stock	
partition_key:	w_id	partition_key:	d_w_id	partition_key:	c_w_id	partition_key:	h_c_id	partition_key:	s_w_id
w_id	w_name	d_w_id	d_id	c_w_id	c_d_id	h_c_w_id	h_c_d_id	s_w_id	s_i_id
	w_street_1		d_name		c_id		h_c_w_id		s_quantity
	w_street_2		d_street_1		c_first		h_d_id		s_dist_01
	w_city		d_street_2		c_middle		h_w_id		s_dist_02
	w_state		d_city		c_last		h_date		s_dist_03
	w_zip		d_state		c_street_1		h_amount		s_dist_04
	w_tax		d_zip		c_street_2		h_data		s_dist_05
	w_ytd		d_tax		c_city	Order line			s_dist_06
Orders			d_ytd		c_state	partition_key:	ol_w_id		s_dist_07
partition_key:	o_w_id		d_next_o_id		c_zip	ol_w_id	ol_d_id		s_dist_08
o_w_id	o_d_id	New orders			c_phone		ol_o_id		s_dist_09
	o_id	partition_key:			c_since		ol_number		s_dist_10
	o_c_id	no_w_id			c_credit		ol_i_id		s_ytd
	o_entry_d				c_credit_lim		ol_supply_w_id		s_order_cnt
	o_carrier_id				c_discount		ol_delivery_d		s_remote_cnt
	o_ol_cnt	Item table is duplicated at all			c_balance		ol_quantity		s_data
	o_all_local	data nodes			c_ytd_payment		ol_amount		
					c_payment_cnt		ol_dist_info		
					c_delivery_cnt				
					c_data				

4.5 60 Day Space Computation

Details of the 60-day space computations along with proof that the database is configured to sustain 8 hours of growth for the dynamic tables (Order, Order-Line, and History) must be disclosed (see Clause 4.2.3).

The details of the 60-day space computations of 2,340 Data nodes are described as follows (Data size are in GB bytes)

Table	Rows	Data	Index	Data-free	5% space	8-H space	Total Space
warehouse	163,215,000.00	17.41	-	-	0.87	-	18.28
district	1,632,150,000.00	201.07	-	228.52	10.05	-	439.64
customer	4,896,450,000,000.00	3,125,766.97	326,731.64	299.36	172,624.93	-	3,625,422.90
new order	1,468,935,000,000.00	42,056.02	-	285.64	2,102.80	-	44,444.46
stock	16,321,500,000,000.00	5,694,008.38	-	285.64	284,700.42	-	5,978,994.44
item	100,000.00	24.09	-	13.71	1.20	-	39.01
history	4,896,450,000,000.00	350,095.08	-	281.07	17,504.75	70,530.01	438,410.92
order	4,896,450,000,000.00	214,569.32	147,148.07	299.36	18,085.87	43,227.05	423,329.65
order_line	48,964,501,163,163.00	4,182,832.27	-	313.07	209,141.61	842,671.66	5,234,958.60
Total		13,609,570.59	473,879.71	2,006.37	704,172.52	956,428.71	15,746,057.90

Free Space	2,006.37
Dynamic Space	4,747,496.66
Static Space	9,335,953.64
Daily Growth	956,428.71
Daily Speed	-

Storage Per Node	31,539.20
Total Storage	73,801,728.00
60-day Space	66,721,676.48
Remaining Space	7,080,051.52

5 Clause 5: Performance Metrics and Response Time Related Items

5.1 Measured tpmC

Measure tpmC must be reported.

The measured tpmC is 2,055,076,649.

5.2 Response Times

Ninetieth percentile and average response times must be reported for all transaction types as well as for the menu response time.

The detailed statistics are reported below (ms).

Transaction Name	Count	Min	Mean Latency	P50 Latency	P90 Latency	Max
new-order	246,609,197,973	101.030	138.450	121.900	163.300	59,999.900
payment	236,407,712,464	100.657	137.156	120.700	168.100	59,999.900
order-status	22,050,605,728	100.800	166.628	156.000	190.200	59,999.900
stock-level	22,065,766,770	101.185	151.218	138.400	175.500	59,999.900
delivery	22,067,995,563	100.090	103.037	100.400	101.500	11,804.000
menu	550,005,018,713	100.071	103.272	100.400	101.400	11,813.200
delivery_deferred	22,088,925,554	2.408	33.936	23.900	61.000	59,999.900

5.3 Keying and Think Times

The minimum, the average, and the maximum keying and think times must be reported for all transaction types.

The detailed statistics are reported below.

Transaction Name	Thinking Time		
	Min (sec.)	Avg (sec.)	Max (sec.)
New-Order	0.000	12.079	122.645
Payment	0.000	12.079	122.497
Order-Status	0.000	10.086	102.293
Stock-Level	0.000	5.096	52.408

Delivery	0.000	5.097	52.285
Keying Time			
Transaction Name	Min (sec.)	Avg (sec.)	Max (sec.)
New-Order	18.001	18.002	19.893
Payment	3.001	3.002	4.883
Order-Status	2.001	2.002	3.819
Stock-Level	2.001	2.002	3.816
Delivery	2.001	2.002	3.827

5.4 Response Time Frequency Distribution Curves

Response Time frequency distribution curves (see Clause 5.6.1) must be reported for each transaction type.

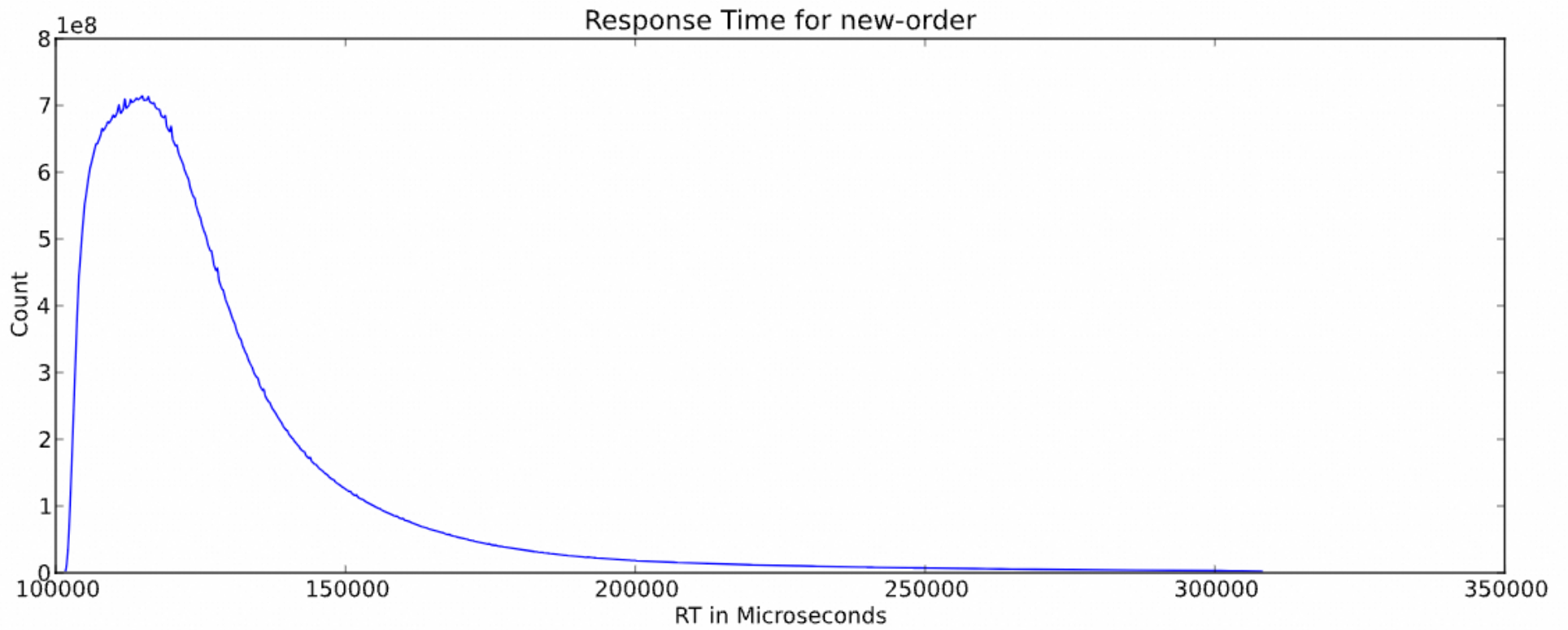


Figure 5. Frequency distribution of response times for New-order

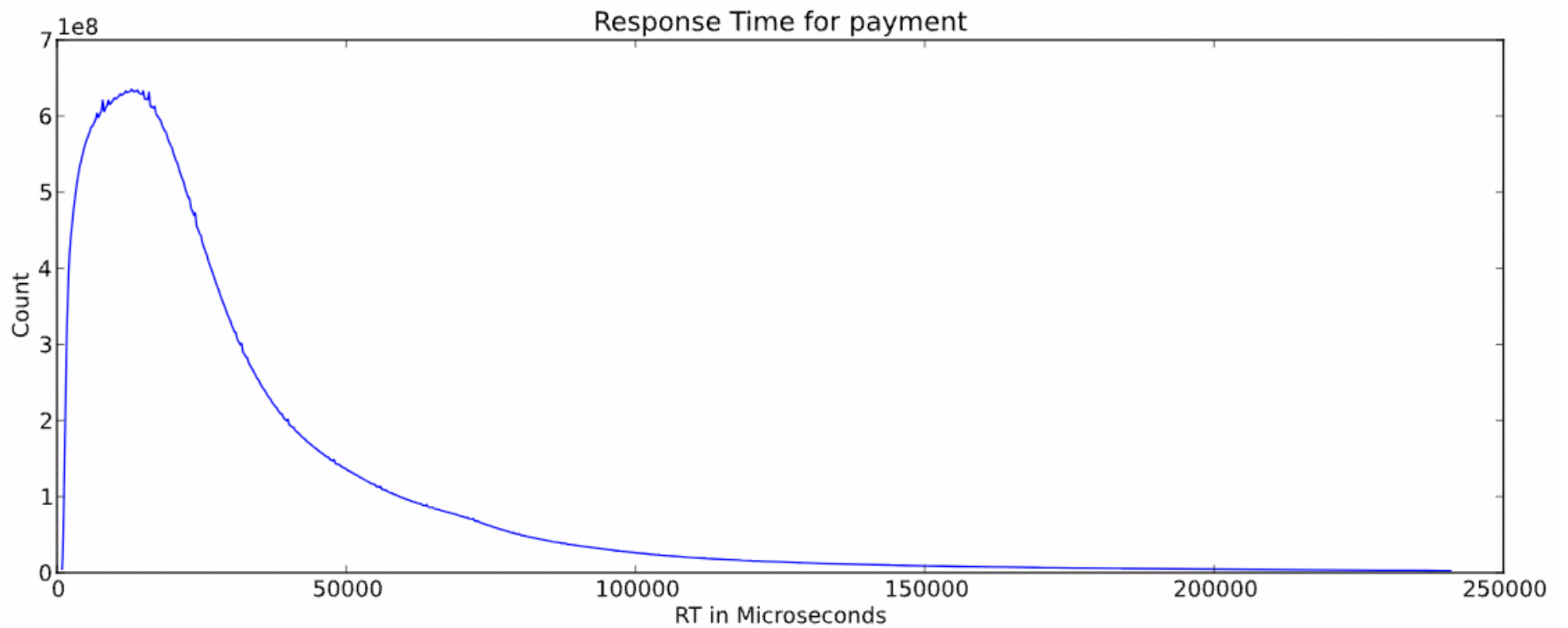


Figure 6. Frequency distribution of response times for Payment

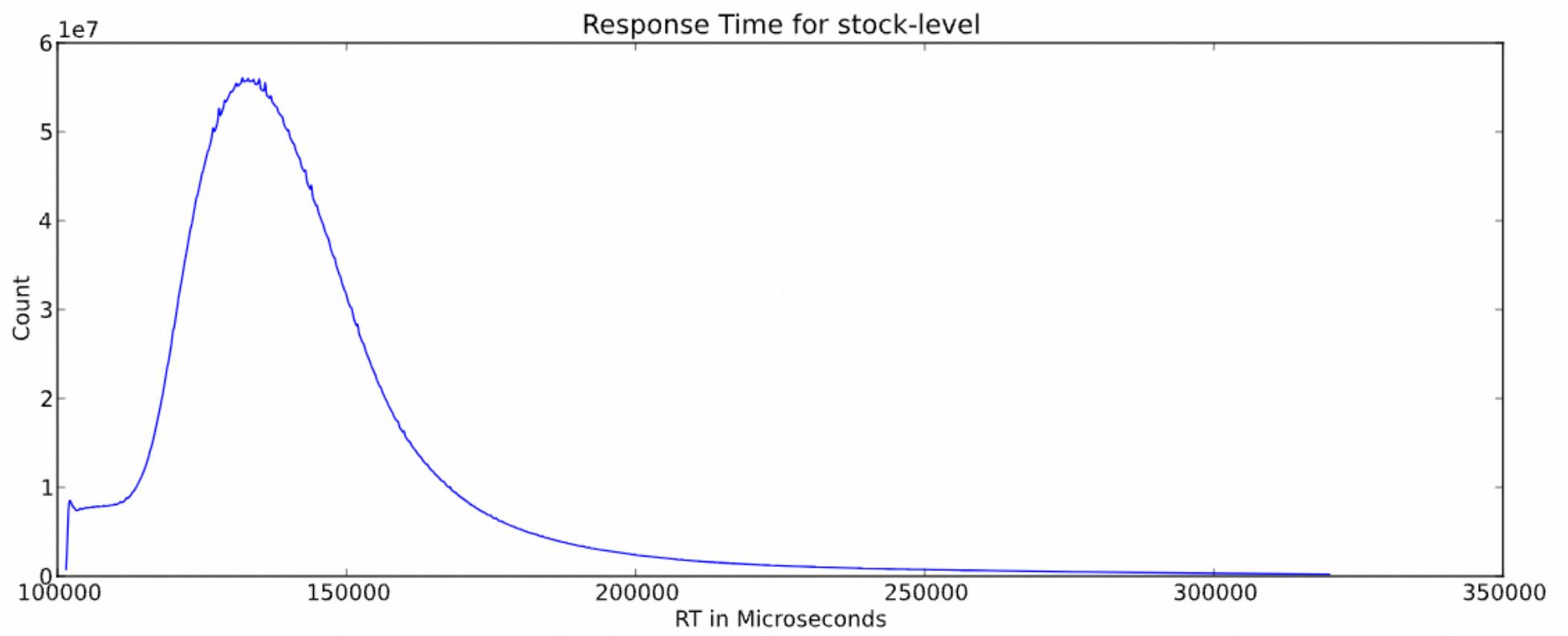


Figure 7. Frequency distribution of response times for Stock-level

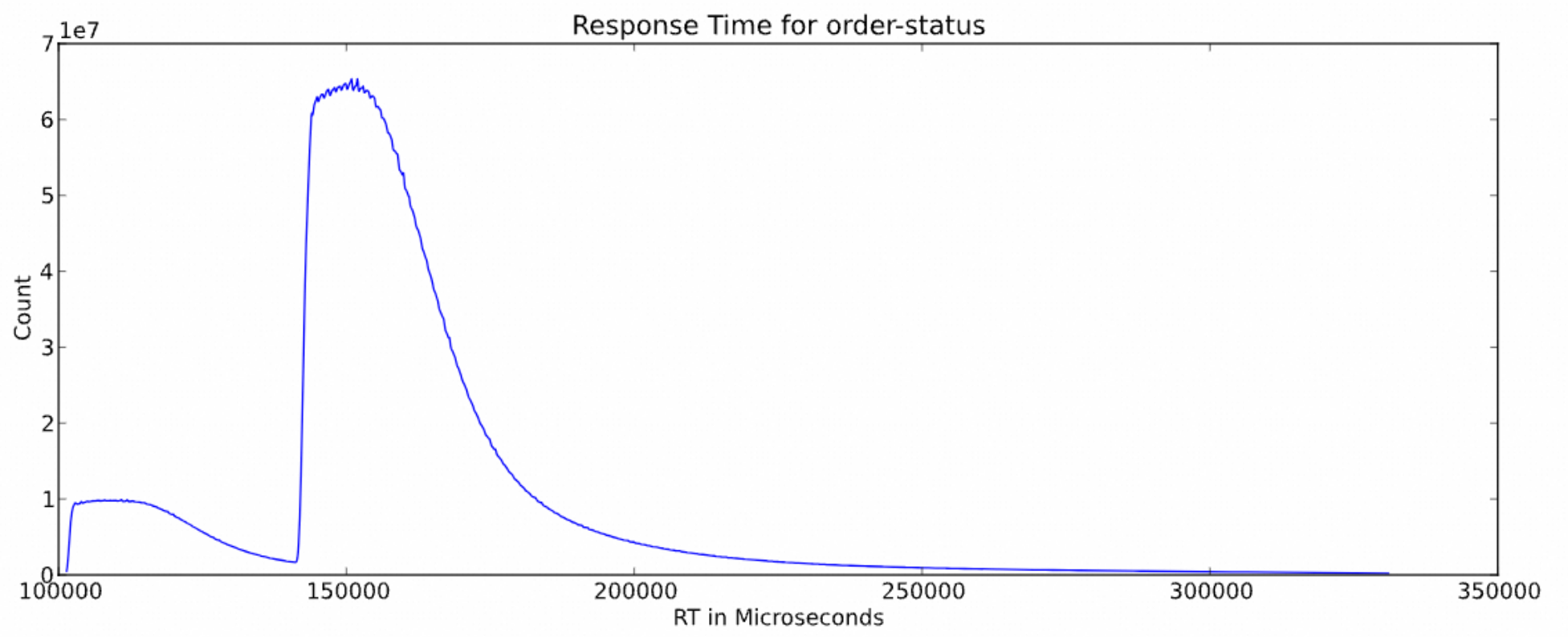


Figure 8. Frequency distribution of response times for Order-status

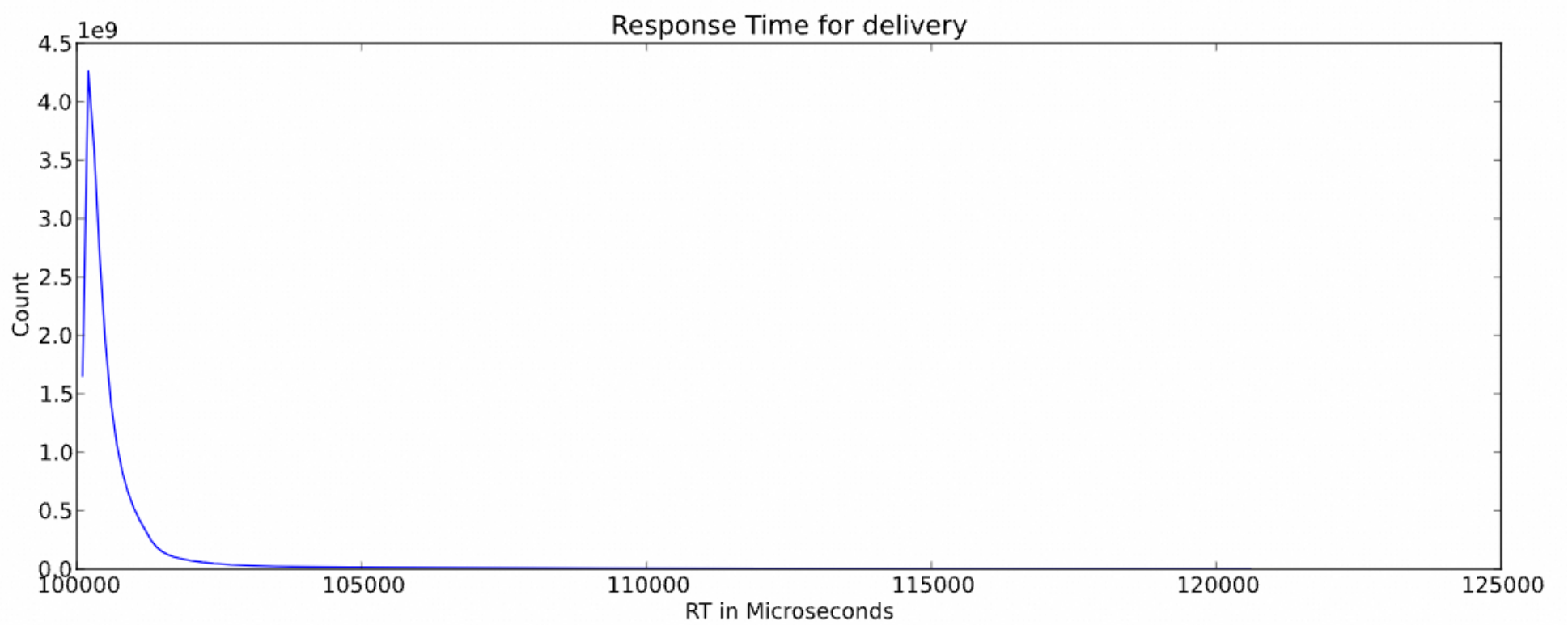


Figure 9. Frequency distribution of response times for Delivery

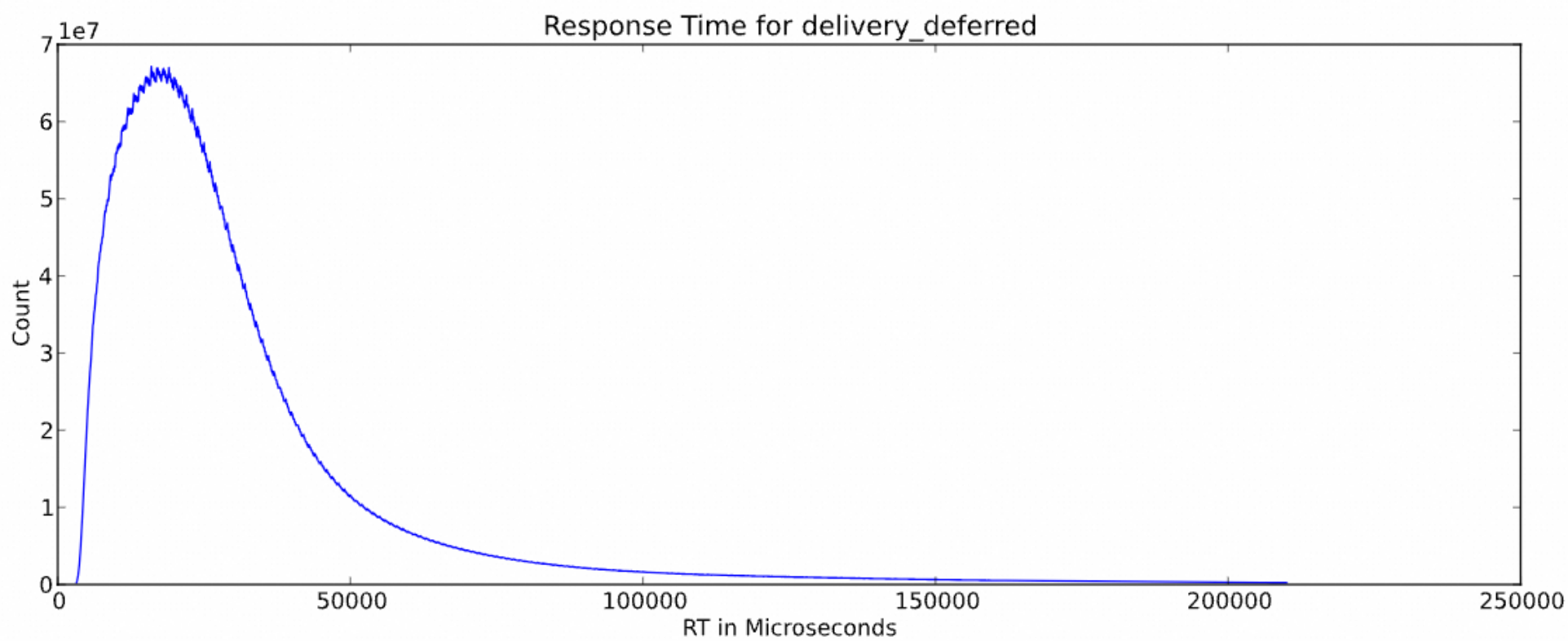


Figure 10. Frequency distribution of response times for Deferred Delivery

5.5 Think Time Frequency Distribution

Think Time frequency distribution curves (see Clause 5.6.3) must be reported for the New-Order transaction.

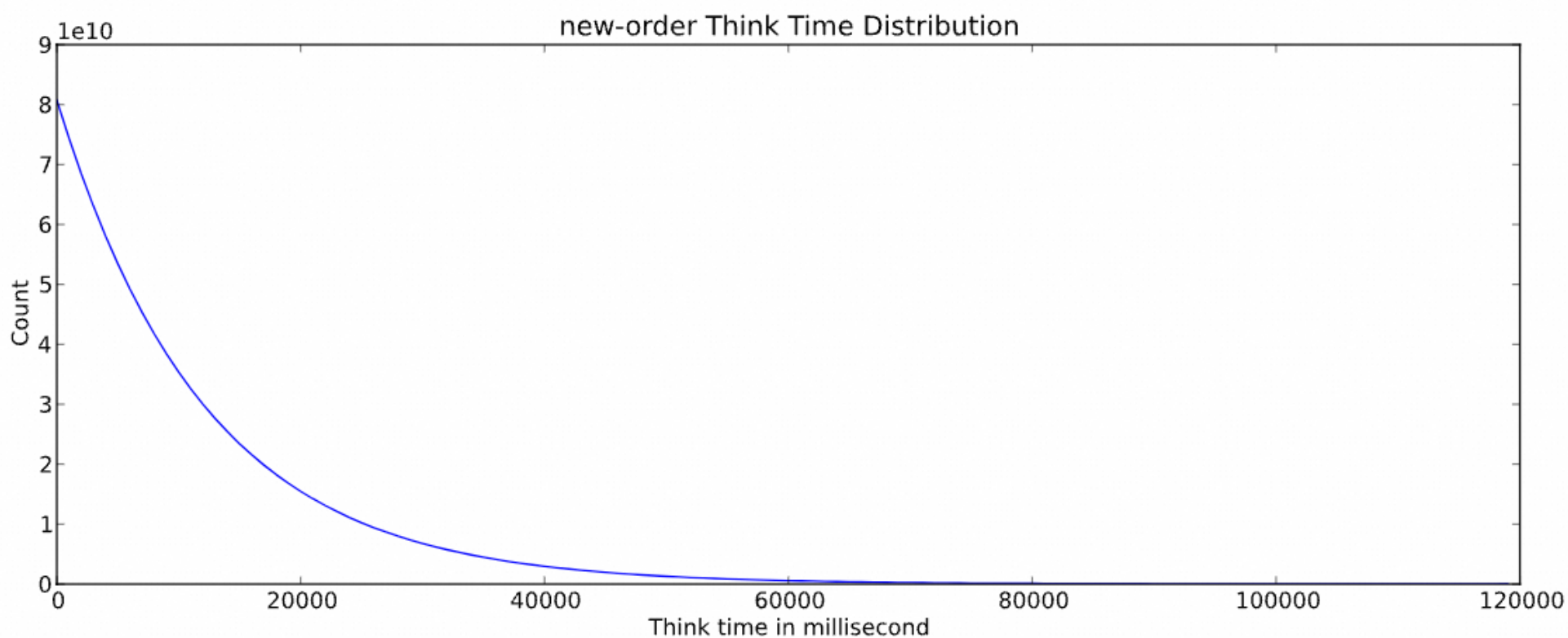


Figure 11. Frequency distribution of thinking times for New-order

5.6 Response Times versus Throughput

The performance curve for response times versus throughput (see Clause 5.6.2) must be reported for the New-Order transaction.

New-order response time vs Throughput

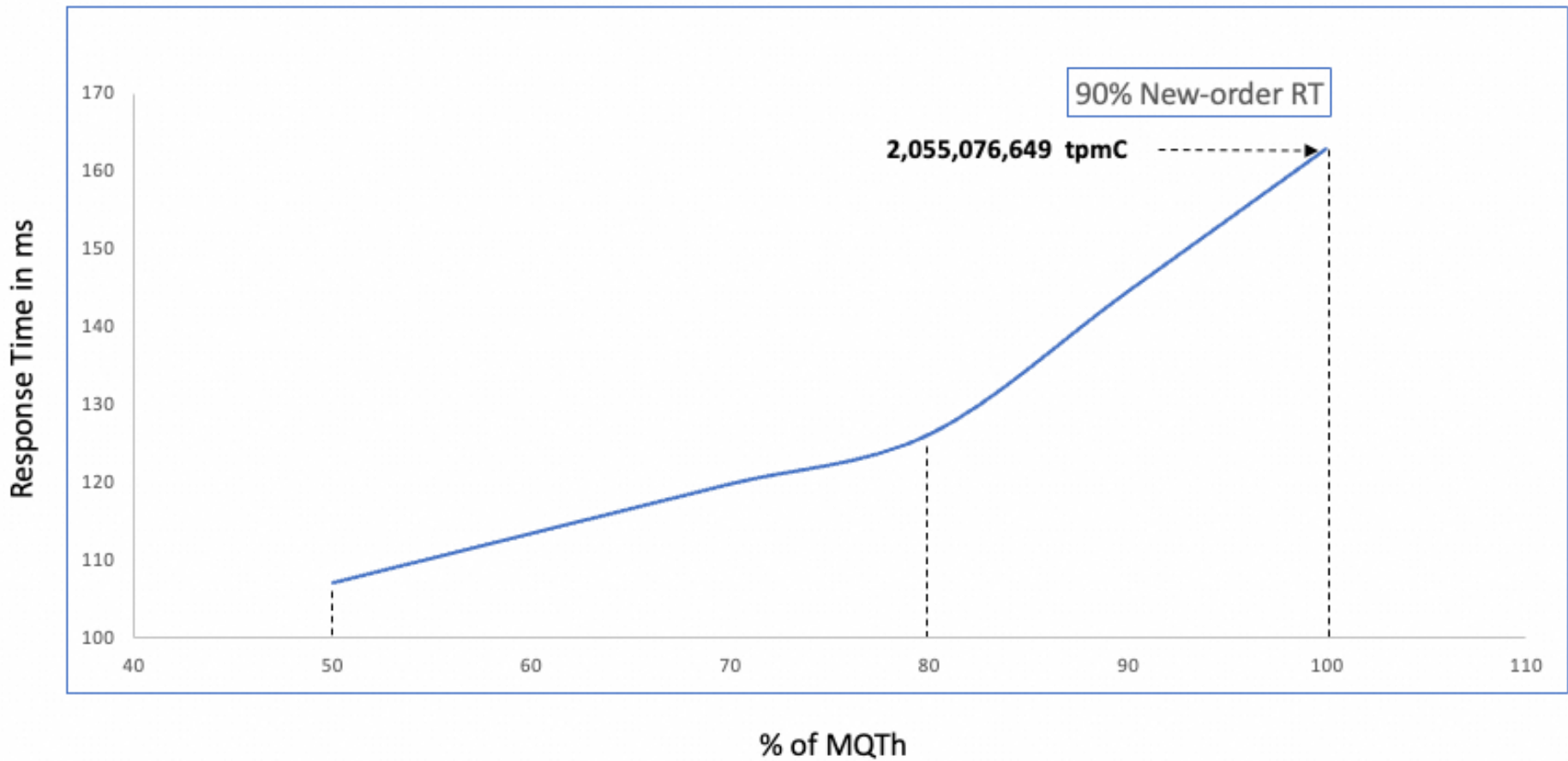


Figure 12 New-order response time versus throughput

5.7 Throughput versus Elapsed Time

A graph of throughput versus elapsed time (see Clause 5.6.4) must be reported for the New-Order transaction.

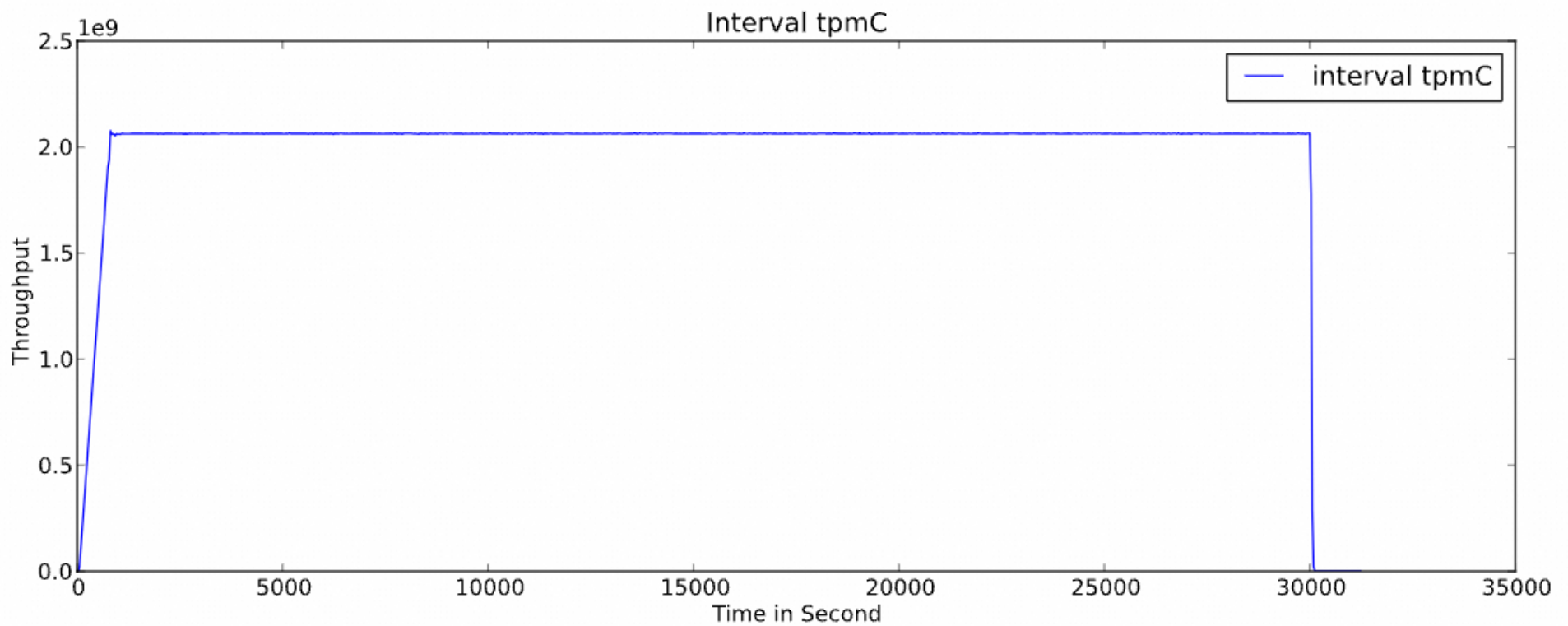


Figure 13 Throughput versus elapsed time

5.8 Steady State Determination

The method used to determine that the SUT had reached a steady state prior to commencing the measurement interval (see Clause 5.5) must be described.

Figure 13 illustrates the New-order throughput over elapsed time. At the start of the benchmark, the system enters a ramp-up phase, lasting approximately 20 minutes, during which throughput gradually increases until it stabilizes. The steady state, used for valid measurement, spans 480 minutes, from 20 minutes to 500 minutes after the benchmark begins. Following this, a ramp-down phase of around 20 minutes occurs, during which throughput gradually decreases to 0 tpmC. Throughout the steady state, the throughput volatility remains under 1%.

5.9 Work Performance During Steady State

A description of how the work normally performed during a sustained test (for example checkpointing, writing redo/undo log records, etc.), actually occurred during the measurement interval must be reported.

During the sustained test period, PolarDB ensures compliance with all ACID properties required by the benchmark specification. The RTE submits transaction requests via HTTP to the web server on the client node. These transactions are selected from one of the five standard transactions outlined in the specification. The RTE simulates user input and decision-making by waiting for a specified period, records the time it submits data to the server, and captures the server's response time. This process is repeated continuously until the test concludes.

On the SUT side, PolarDB Limitless processes the transactions and executes queries sequentially within each transaction, maintaining all ACID properties. During transaction commits, the redo logs are synchronized with standby replicas through PolarDB's physical replication mechanism. A transaction is marked as committed only after receiving an acknowledgment (ACK) from the standby replica. For this test, the checkpoint interval in PolarDB is configured to 1 second (). If the RW node fails, the standby node can seamlessly take over as the new RW node to ensure uninterrupted service.

5.10 Measurement Interval

A statement of the duration of the measurement interval for the reported Maximum Qualified Throughput (tpmC) must be included.

A precise duration of 2 hours (2 hours = 120 minutes = 7,200 seconds) is designated as the measurement interval for reporting the steady performance of Maximum Qualified Throughput (tpmC).

5.11 Transaction Mix Regulation

The method of regulation of the transaction mix (e.g., card decks or weighted random distribution) must be described. If weighted distribution is used and the RTE adjusts the weights associated with each transaction type, the maximum adjustments to the weight from the initial value must be disclosed.

The weighted distribution algorithm in the RTE manages the transaction mix percentage to align with the specification requirements. The transaction weights were predetermined before the test and remained constant throughout its duration.

5.12 Transaction Mix

The percentage of the total mix for each transaction type must be disclosed.

The detailed statistics are reported below

Transaction Mix
New-Order:44.903%
Payment:43.046%
Order-Status:4.015%
Delivery:4.018%
Stock-Level:4.018%

5.13 Percentage of New-Order Transactions

The percentage of New-Order transactions rolled back as a result of invalid item number must be disclosed.

The detailed statistics are reported below

NewOrder Txn
Percentage of Home order-lines: 99.048%
Percentage of Remote order-lines: 0.952%
Percentage of Rolled Back Transactions:1.004%
Avg. Number of items per Transactions:10.00

5.14 Number of Order-lines per New-Order

The average number of order-lines entered per New-Order transaction must be disclosed.

The detailed statistics are reported below

NewOrder Txn
Percentage of Home order-lines: 99.048%

Percentage of Remote order-lines: 0.952%
Percentage of Rolled Back Transactions:1.004%
Avg. Number of items per Transactions:10.00

5.15 Percentage of Remote Order-lines per New-Order

The percentage of remote order-lines entered per New-Order transaction must be disclosed.

a) Percentage of Remote Payments

The percentage of remote payment transactions must be disclosed.

b) Percentage of access customer by C_LAST for Payment and Order-Status

The percentage of customer selections by customer last name in the Payment and Order-Status transactions must be disclosed.

c) Percentage of Skipped Delivery Transactions

The percentage of Delivery transactions skipped due to there being fewer than necessary orders in the New-Order table must be disclosed.

d) Checkpoints

The number of checkpoints in the Measurement Interval, the time in seconds from the start of the Measurement Interval to the first checkpoint and the Checkpoint Interval must be disclosed

The detailed statistics are reported below.

NewOrder Txn
Percentage of Home order-lines: 99.048%
Percentage of Remote order-lines: 0.952%
Percentage of Rolled Back Transactions:1.004%
Avg. Number of items per Transactions:10.00

5.16 Percentage of Remote Payments

The percentage of remote payment transactions must be disclosed.

The detailed statistics are reported below.

Payment Txn
Percentage of Home Transactions:85.000%
Percentage of Remote Transactions:15.000%

5.17 Percentage of access customer by C_LAST for Payment and Order-Status

The percentage of customer selections by customer last name in the Payment and Order-Status transactions must be disclosed.

The detailed statistics are reported below.

Access by C_LAST (Non-primary key)
Percentage of Payment Transactions:60.000%
Percentage of Order-Status Transactions:60.000%

5.18 Percentage of Skipped Delivery Transactions

The percentage of Delivery transactions skipped due to there being fewer than necessary orders in the NewOrder table must be disclosed.

The detailed statistics are reported below.

Delivery Txn
Skipped transactions:0.00
Percentage of Skipped transactions:0.000%

5.19 Checkpoints

The number of checkpoints in the Measurement Interval, the time in seconds from the start of the Measurement Interval to the first checkpoint and the Checkpoint Interval must be disclosed.

During the measurement interval, each DB node performed an average of 291,275 checkpoints. The number of checkpoints per node ranged from a minimum of 125,648 to a maximum of 538,640. The checkpoint interval across all nodes was 181 ms on average, each checkpoint interval ranged from a minimum of 1 ms to a maximum of 74,696 ms. The checkpoint history is derived from the checkpoint.info file on each node, and each database node independently performs checkpoints.

The server appends redo logs instead of overwriting them in cycles, theoretically providing unlimited log space. A background thread periodically advances the checkpoint lazily, controlled by the “innodb_log_checkpoint_every” setting, which is currently configured to 1 second. Additionally, a soft limit can be set for uncheckpointed log space. If this limit is reached, the background page cleaner threads perform more aggressive page flushing to advance the checkpoint. This is managed using the “innodb_log_max_checkpoint_files” configuration. The maximum amount of uncheckpointed log space is capped at 8 log files (“innodb_polar_log_max_checkpoint_files = 8”), with each log file sized at 1GB (“innodb_log_file_size = 1GB”).

6 Clause 6: SUT, Driver and Communications Related Items

6.1 RTE Description

If the RTE is commercially available, then its inputs must be specified. Otherwise, a description must be supplied of what inputs (e.g. scripts) to the RTE had been used.

The RTE module and the SUT's web request processing are both implemented using the Go programming language. Each RTE machine initiates processes and maintains persistent HTTP connections with Nginx servers. The RTE handles multiple tasks, starting with generating random parameter values that comply with specification requirements, such as keying and think time. It then sets up the simulation model and initializes requests to interact with the SUT. Additionally, the RTE collects key statistics, including response time and the completion or abort status of requests. The RTE operates throughout the entire test duration, encompassing the ramp-up phase, measurement interval, and ramp-down period. Table 7 outlines the main parameter configurations of the RTE.

Table 7 Parameter configuration of RTE

Parameter	Value	Description
warehouse-count	163,215,000	current warehouse number
http-conn-count	1,632,150,000	simulating client number
RTE nodes/instances	558	RTE node/instance number
data-path	./results/data	path of statistical data
ramp-up-time	20 minutes	warmup time
ramp-down-time	20 minutes	ramp down time
measurement-interval	2 hours	time of steady running
print-interval	30 seconds	print interval, less than 0s will not print

6.2 Number of Terminal Connection Lost

The number of terminal connections lost during the Measurement Interval must be disclosed (see Clause 6.6.2).

The terminal connection remains active throughout the measurement interval.

6.3 Emulated Components

It must be demonstrated that the functionality and performance of the components being emulated in the Driver System are equivalent to that of the priced system. The results of the test described in Clause 6.6.3.4 must be disclosed.

The SUT delivers services to the emulated system via HTTP. Each RTE machine initiates 100 processes (), with each process maintaining 58,500 active HTTP connections to the SUT's Nginx servers.

6.5 Network Configuration

The network configurations of both the tested services and the proposed (target) services, which are being represented, and a thorough explanation of exactly which parts of the proposed configuration are being replaced with the Driver System must be disclosed (see Clause 6.6.4).

All machines are located within a single network zone, featuring an end-to-end speed of 25 Gbps.

6.6 Operator Intervention

If the configuration requires operator intervention, the mechanism and the frequency of this intervention must be disclosed.

No human intervention occurs during the entire measurement interval. PolarDB Limitless operates autonomously, leveraging intelligent distribution to ensure consistent performance and high transactional throughput.

7 Clause 7: Pricing Related Items

7.1 Hardware and software Price

A detailed list of hardware and software used in the priced system must be reported. Each separately orderable item must have vendor part number, description, release/revision level, and either general availability status or committed delivery date. If package pricing is used, vendor part number of the package and a description uniquely identifying each of the components of the package must be disclosed. Pricing source(s) and effective date(s) of price(s) must also be reported

A detailed list of the hardware and software used in the priced system is included in the Executive Summary, with pricing information provided in Appendix D.

7.2 Total 3-Year Cost

The total 3-year price of the entire Priced Configuration must be reported, including: hardware, software, and maintenance charges. The justification of any Discounts applied must be disclosed in the price sheet. Sufficient detail of what items are being discounted and by how much they are being discounted must be provided so that the Discount amount used in the computation of the total system cost can be independently reproduced.

The total 3-year cost of the complete Priced Configuration is included in the Executive Summary.

7.3 Availability Date

The Committed delivery date for general availability (availability date) of products used in the price calculations must be reported. The Availability Date must be reported on the first page of the Executive Summary and with a precision of one day. When the priced system includes products with different availability dates, the reported availability date for the priced system must be the date at which all Components are committed to be Generally Available. Each Component used in the Priced Configuration is considered to be Available on the Availability Date unless an earlier date is specified.

All products included in the price calculation are available as of January 19, 2025.

7.4 Hardware and Software Support

Alibaba Cloud offers enterprise-level client services, including 24/7 phone support for all VM instances and physical machines. The PolarDB team provides professional support for database systems, also available 24/7 via phone.

7.5 Statement of measured tpmC and Price/Performance

The measured tpmC, along with the corresponding calculations for 3-year pricing, price/performance (price/tpmC), and the availability date, must be provided.

Table 8 Statement of tpmC and price/performance

Total System Cost	TPC-C® Throughput	Price/Performance	Availability Date
Three-year cost of hardware, software, and maintenance	Maximum Qualified Throughput by transaction per minute - C (tpmC)	Total System Cost/tpmC	Date for all hardware, software, and maintenance available to purchase
1,626,643,837.00 CNY	2,055,076,649	0.80 CNY	Jan 19th 2025

7.6 Country Specific Pricing

Additional Clause 7 related items may be included in the Full Disclosure Report for each country specific priced configuration. Country specific pricing is subject to Clause 7.1.7.

All components of the Priced Configuration are priced in Renminbi (RMB) or Chinese Yuan (CNY), the official currency of the People's Republic of China.

7.7 Orderability Date

For each of the components that are not orderable on the report date of the FDR, the following information must be included in the FDR:

- Name and part number of the item that is not orderable
- The date when the component can be ordered (on or before the Availability Date)
- The method to be used to order the component (at or below the quoted price) when that date arrives
- The method for verifying the price

All components of the Priced Configuration are available as of January 19, 2025.

8 Clause 8: Auditor Attestation

8.1 Auditor Information

The auditor's name, address, phone number, and a copy of the auditor's attestation letter indicating compliance must be included in the Full Disclosure Report

This benchmark was audited by:
PerfLabs, Inc. DBA InfoSizing
Doug Johnson
63 Lourdes Drive Leominster, MA 01453, USA
Phone: +1 (978) 343-6562
www.sizing.com

8.2 Attestation Letter

The Auditor's Attestation Letter is included in the following pages.

Panfeng Zhou, Xinjun Yang, Feifei Li
Database Products Business Unit
Alibaba Cloud Computing Co., Ltd.
Yungu Park, No. 100B Dengcai Street
Hangzhou China

January 27, 2025

I verified the TPC Benchmark™ C v5.11.0 performance of the following configuration:

Platform: Alibaba Cloud PolarDB Limitless (with 2,340 PolarDB for MySQL Data Nodes)
Operating System: Alibaba Group Enterprise Linux Server 7.2 (Paladin)
Database Manager: Alibaba Cloud PolarDB for My SQL (PolarDB-M) 8.0.2 Enterprise Edition

The results were:

Performance Metric 2,055,076,649 tpmC
Number of Users 1,632,150,000

Server	Alibaba Cloud PolarDB Limitless		
Nodes	2,340 DB Nodes, 3 Coordinator Nodes, 558 Client Nodes		
CPU	2x Intel® Xeon® Platinum 8575C (3.20 GHz, 48-core, 320 MB Cache) (DB Node Servers) 2x Intel® Xeon® Platinum 8269CY CPU (2.50 GHz, 26-Core, 35.75 MB Cache) (Coordinator/Client Servers)		
Memory	2,048 GB (DB Node Servers) 768 GB (Coordinator/Client Servers)		
Storage	Qty	Size	Type
	1	240 GB	OS disk (All Servers)
	14	9.6 TB	NVMe (DB Node Servers)

In my opinion, these performance results were produced in compliance with the TPC requirements for the benchmark.

The following verification items were given special attention:

- The transactions were correctly implemented.
- The database records were the proper size.
- The database was properly scaled and populated.
- The ACID properties were met.
- Input data was generated according to the specified percentages.

63 Lourdes Dr. | Leominster, MA 01453 | 978-343-6562 | www.sizing.com

- The transaction cycle times included the required keying and think times.
- The reported response times were correctly measured.
- At least 90% of all delivery transactions met the 80 Second completion time limit.
- All the 90% response times were below the specified maximums.
- The measurement interval was representative of steady state conditions.
- The reported measurement interval was at least 120 minutes.
- Checkpoint intervals were under 30 minutes.
- The 60-day storage requirement was correctly computed.
- The system pricing was verified for major components and maintenance.

Additional Audit Notes:

None.

Respectfully Yours,



Doug Johnson, Certified TPC Auditor

63 Lourdes Dr. | Leominster, MA 01453 | 978-343-6562 | www.sizing.com

Appendix A: Source Code File List

The source code and scripts used to implement the benchmark are provided as a soft appendix. This soft appendix includes the following files:

```
.
|-- Makefile
|-- README.md
|-- check
|-- cmd
|   |-- check.go
|   |-- ctrl.go
|   `-- main.go
|-- config.yaml
|-- ctrl
|-- go.mod
|-- go.sum
|-- main
|-- openresty
|   |-- README.md
|   |-- build.sh
|   |-- html
|   |   |-- index.html
|   |   |-- input
|   |   |   |-- delivery.html
|   |   |   |-- new-order.html
|   |   |   |-- order-status.html
|   |   |   |-- payment.html
|   |   |   `-- stock-level.html
|   |   |-- menu.html
|   |   |-- script.js
|   |   `-- theme.css
|   |-- install.sh
|   |-- lua
|   |   |-- common.lua
|   |   |-- delivery.lua
|   |   |-- delivery_processor.lua
|   |   |-- new-order.lua
|   |   |-- order-status.lua
|   |   |-- payment.lua
|   |   |-- queue.lua
|   |   `-- stock-level.lua
|   |-- luarocks-3.3.1.tar.gz
|   |-- nginx.conf.template
|   |-- openresty-1.17.8.1.tar.gz
|   |-- openresty-1.19.9.1.tar.gz
|   |-- openresty-1.25.3.1.tar.gz
|   |-- start.sh
|   |-- stop.sh
|   `-- test.sh
|-- pkg
|   |-- acid
|   |   |-- Makefile
|   |   |-- atomic_test.go
|   |   |-- config.yaml
|   |   |-- consistency_test.go
|   |   |-- durability_test.go
|   |   |-- isolation_procedure.sql
|   |   `-- isolation_test.go
|   |-- client
|   |   |-- client.go
|   |   |-- emulate_context.go
|   |   `-- terminal.go
|   |-- collection
|   |   |-- checkpoint.go
|   |   |-- config.go
|   |   |-- directory.go
|   |   |-- disk_size.go
```

```

| | |-- log.go
| | |-- ssh.go
| | |-- system_info.go
| | |-- table_size.go
| | |-- tablespace.go
| | |-- version.go
| | `-- wal_size.go
|-- common
|   |-- common.go
|-- config
|   |-- config.go
|   |-- config_test.go
|   `-- remote_worker.go
|-- connection
|   |-- conn.go
|-- hist
|   |-- histogram.go
|-- loader|   |-- check.go
|   |-- config_store.go
|   |-- ddl.go
|   `-- loader.go
|-- metric
|   |-- metric.go
|-- txn
|   |-- tpcc_delivery.go
|   |-- tpcc_neworder.go
|   |-- tpcc_orderstatus.go
|   |-- tpcc_payment.go
|   |-- tpcc_random.go
|   |-- tpcc_split_result.go
|   |-- tpcc_stocklevel.go
|   |-- tpcc_txn.go
|   `-- tpcc_txn_test.go
`-- utils
    |-- dump_stack.go
    |-- run_sql.go
    `-- run_sql_test.go
-- report_scripts
| |-- generateReport.sh
| |-- rt_plot.py
| |-- tTime_plot.py
| `-- tpm_plot.py
-- sql
    |-- buildFinish.sql
    |-- data_distribution.sql
    |-- dbCheck.sql
    |-- indexCreates.sql
    |-- indexDrops.sql
    |-- preCheckRunInit.sql
    |-- preLoadRunInit.sql
    |-- samplingCheck.sql
    |-- storedProcedureCreates.sql
    |-- storedProcedureDrops.sql
    |-- tableCheck.sql
    |-- tableCreates.sql
    |-- tableDrops.sql
    `-- tableTruncates.sql

```

Total number of files: 101
Total number of folders: 19

Appendix B: Database design

The schema of the database system is provided as follows.

B.1 Table creation

```
CREATE TABLE bmsql_warehouse (  
  w_id INT NOT NULL,  
  w_name VARCHAR(10),  
  w_street_1 VARCHAR(20),  
  w_street_2 VARCHAR(20),  
  w_city VARCHAR(20),  
  w_state CHAR(2),  
  w_zip CHAR(9),  
  w_tax DECIMAL(4, 4),  
  w_ytd DECIMAL(12, 2),  
  PRIMARY KEY (w_id)  
) tbdistribution by hash(w_id) tbdistributions %d;
```

```
CREATE TABLE bmsql_district (  
  d_w_id INT NOT NULL,  
  d_id TINYINT NOT NULL,  
  d_name VARCHAR(10),  
  d_street_1 VARCHAR(20),  
  d_street_2 VARCHAR(20),  
  d_city VARCHAR(20),  
  d_state CHAR(2),  
  d_zip CHAR(9),  
  d_tax DECIMAL(4, 4),  
  d_ytd DECIMAL(12, 2),  
  d_next_o_id INT,  
  PRIMARY KEY (d_w_id, d_id)  
) tbdistribution by hash(d_w_id) tbdistributions %d;
```

```
CREATE TABLE bmsql_customer (  
  c_w_id INT NOT NULL,  
  c_d_id TINYINT NOT NULL,  
  c_id INT NOT NULL,  
  c_first VARCHAR(16) NOT NULL,  
  c_middle CHAR(2),  
  c_last VARCHAR(16) NOT NULL,  
  c_street_1 VARCHAR(20),  
  c_street_2 VARCHAR(20),  
  c_city VARCHAR(20),  
  c_state CHAR(2),  
  c_zip CHAR(9),  
  c_phone CHAR(16),  
  c_since DATETIME(3),  
  c_credit CHAR(2),  
  c_credit_lim DECIMAL(12, 2),  
  c_discount DECIMAL(4, 4),  
  c_balance DECIMAL(12, 2),  
  c_ytd_payment DECIMAL(12, 2),  
  c_payment_cnt SMALLINT,  
  c_delivery_cnt INT,  
  c_data TEXT,  
  PRIMARY KEY(c_w_id, c_d_id, c_id)  
) tbdistribution by hash(c_w_id) tbdistributions %d;
```

```
CREATE TABLE bmsql_history (  
  h_c_id INT,  
  h_c_d_id TINYINT,  
  h_c_w_id INT,  
  h_d_id TINYINT,  
  h_w_id INT,  
  h_date DATETIME(3),  
  h_amount DECIMAL(6, 2),  
  h_data VARCHAR(24)  
) tbdistribution by hash(h_c_w_id) tbdistributions %d;
```



```

CREATE TABLE bmsql_new_order (
  no_w_id INT NOT NULL,
  no_d_id TINYINT NOT NULL,
  no_o_id INT NOT NULL,
  PRIMARY KEY(no_w_id, no_d_id, no_o_id)
) tbdistribution by hash(no_w_id) tbdistributions %d;

```

```

CREATE TABLE bmsql_oorder (
  o_w_id INT NOT NULL,
  o_d_id TINYINT NOT NULL,
  o_id INT NOT NULL,
  o_c_id INT NOT NULL,
  o_entry_d DATETIME(3),
  o_carrier_id TINYINT,
  o_ol_cnt TINYINT,
  o_all_local TINYINT,
  PRIMARY KEY(o_w_id , o_d_id , o_id)
) tbdistribution by hash(o_w_id) tbdistributions %d;

```

```

CREATE TABLE bmsql_order_line (
  ol_w_id INT NOT NULL,
  ol_d_id TINYINT NOT NULL,
  ol_o_id INT NOT NULL,
  ol_number TINYINT NOT NULL,
  ol_i_id INT,
  ol_supply_w_id INT,
  ol_delivery_d DATETIME(3),
  ol_quantity TINYINT,
  ol_amount DECIMAL(6 , 2 ),
  ol_dist_info CHAR(24),
  PRIMARY KEY(ol_w_id , ol_d_id , ol_o_id , ol_number)
) tbdistribution by hash(ol_w_id) tbdistributions %d;

```

```

CREATE TABLE bmsql_stock (
  s_w_id INT NOT NULL,
  s_i_id int NOT NULL,
  s_quantity SMALLINT,
  s_dist_01 CHAR(24),
  s_dist_02 CHAR(24),
  s_dist_03 CHAR(24),
  s_dist_04 CHAR(24),
  s_dist_05 CHAR(24),
  s_dist_06 CHAR(24),
  s_dist_07 CHAR(24),
  s_dist_08 CHAR(24),
  s_dist_09 CHAR(24),
  s_dist_10 CHAR(24),
  s_ytd DECIMAL(8 , 0 ),
  s_order_cnt SMALLINT,
  s_remote_cnt SMALLINT,
  s_data VARCHAR(50),
  PRIMARY KEY(s_w_id, s_i_id)
) tbdistribution by hash(s_w_id) tbdistributions %d;

```

```

CREATE TABLE bmsql_item (
  i_id INT NOT NULL,
  i_im_id INT,
  i_name VARCHAR(24),
  i_price DECIMAL(5 , 2 ),
  i_data VARCHAR(50),
  PRIMARY KEY(i_id)
) global_table=1;

```

B.2 Procedure creation

```

DROP PROCEDURE IF EXISTS bmsql_proc_new_order;
delimiter $$
CREATE PROCEDURE bmsql_proc_new_order(
  in in_w_id int,
  in in_d_id int,
  in in_c_id int,
  in in_o_ol_cnt int,
  in in_o_all_local int,
  in in_ol_i_id_array varchar(512),      -- The item numbers(item_1,...,item_cnt)
  in in_ol_supply_w_id_array varchar(512), -- The supplying warehouse numbers(warehouse_1,...,warehouse_cnt)
  in in_ol_quantity varchar(512))      -- The item quantities(q1,...,q_cnt)
label:BEGIN

  -- The variables which will be returned
  declare err int;
  declare out_o_entry_d char(81);        -- Date time
  declare out_w_tax decimal(4,2);       -- Warehouse tax
  declare out_d_tax decimal(4,2);       -- District tax
  declare out_o_id int;                  -- District next order id
  declare out_c_last varchar(16);       -- Customer last name
  declare out_c_discount decimal(4,2);  -- Customer's discount rate
  declare out_c_credit char(2);         -- Customer's credit status
  declare out_total_amount decimal(8,2); -- Total amount

  -- Integrated variables which will be returned
  declare out_ol_supply_w_ids varchar(512); -- Warehouse ids of supplier
  declare out_i_ids varchar(512);          -- Item ids
  declare out_ol_quantities varchar(512);  -- The quantity of each item
  declare out_item_names varchar(512);     -- Item names
  declare out_supply_quantities varchar(512); -- Item supplying quantities
  declare out_brand_generic varchar(64);   -- Item brand generic info
  declare out_prices varchar(512);        -- Item price
  declare out_amounts varchar(512);       -- Total amount of an item

  -- The variables used in procedure
  declare var_i_price decimal(5,2) default NULL; -- Item price
  declare var_i_name varchar(24) default NULL;  -- Item name
  declare var_i_data varchar(50) default NULL;  -- Item data
  declare var_s_quantity smallint default NULL; -- Supplying quantity
  declare var_s_data varchar(50) default NULL;  -- Supplying data
  declare var_ol_dist_info varchar(24) default NULL; -- The order line district info
  declare var_bg varchar(2) default NULL;       -- Brand generic info
  declare var_ol_amount decimal(6,2) default NULL; -- Order line amount
  declare var_ol_supply_w_id int default NULL;  -- Item warehouse
  declare var_ol_i_id int default NULL;        -- The order line item id
  declare var_ol_quantity int default NULL;    -- The order line item quantity
  declare var_remote int default NULL;         -- Whether the item is supplied by remote
  declare var_ol_number int default 1;        -- The order line number

  -- defining error handling
  -- declare continue HANDLER FOR SQLWARNING,NOT FOUND,SQLEXCEPTION
  declare exit HANDLER FOR NOT FOUND,SQLEXCEPTION
BEGIN
  IF err = 1 THEN
    -- Cache an error when it gets a invalid item id.
    SET out_ol_supply_w_ids = concat_ws(',', out_ol_supply_w_ids, var_ol_supply_w_id);
    SET out_i_ids = concat_ws(',', out_i_ids, var_ol_i_id);
    SET out_ol_quantities = concat_ws(',', out_ol_quantities, var_ol_quantity);
    SET out_item_names = concat_ws(',', out_item_names, 'NO ITEM');
    SET out_supply_quantities = concat_ws(',', out_supply_quantities, 0);
    SET out_brand_generic = concat_ws(',', out_brand_generic, ' ');
    SET out_prices = concat_ws(',', out_prices, 0);
    SET out_amounts = concat_ws(',', out_amounts, 0);
  END IF;

  SET err = -1;
  SELECT err,out_ol_supply_w_ids, out_i_ids, out_ol_quantities, out_item_names, out_supply_quantities,
    out_brand_generic, out_prices, out_amounts,out_total_amount,out_c_credit,out_c_discount,out_c_last,
    out_o_id,out_d_tax,out_o_entry_d,out_w_tax;
  ROLLBACK;
END;

```

```

SELECT CURRENT_TIMESTAMP(3) INTO out_o_entry_d;

START TRANSACTION;
SET err = 0;
SET out_total_amount=0;

-- query the user's discount, tax and so on
SELECT c_discount, c_last, c_credit, w_tax INTO out_c_discount, out_c_last, out_c_credit, out_w_tax
FROM bmsql_customer, bmsql_warehouse
WHERE w_id = in_w_id AND c_w_id = w_id AND c_d_id = in_d_id AND c_id = in_c_id;

-- update the order-id and tax of the district
UPDATE bmsql_district
SET d_next_o_id = d_next_o_id + 1
WHERE d_id = in_d_id AND d_w_id = in_w_id
POLAR_RETURNING d_next_o_id - 1, d_tax into out_o_id, out_d_tax;

-- create order
INSERT INTO bmsql_oorder(o_id, o_d_id, o_w_id, o_c_id, o_entry_d, o_ol_cnt, o_all_local)
VALUES (out_o_id, in_d_id, in_w_id, in_c_id, out_o_entry_d, in_o_ol_cnt, in_o_all_local);

-- create unshipped orders
INSERT INTO bmsql_new_order (no_o_id, no_d_id, no_w_id)
VALUES (out_o_id, in_d_id, in_w_id);

-- circulation processing for every item
WHILE var_ol_number <= in_o_ol_cnt DO
-- decoding warehouse, item-id, item-qty
SET var_ol_supply_w_id= substring_index(substring_index(in_ol_supply_w_id_array, ',', var_ol_number), ',', -1);
SET var_ol_i_id= substring_index(substring_index(in_ol_i_id_array, ',', var_ol_number), ',', -1);
SET var_ol_quantity= substring_index(substring_index(in_ol_quantity, ',', var_ol_number), ',', -1);
SET var_remote = 0;
IF var_ol_supply_w_id != in_w_id THEN
SET var_remote = 1;
END IF;

-- Tell the handler that we are searching for item data
SET err = 1;

-- if the number of affected rows is not 1, the next item execution is processed
-- get stock info
UPDATE bmsql_stock
SET s_order_cnt = s_order_cnt + 1, s_ytd = s_ytd + var_ol_quantity, s_remote_cnt = s_remote_cnt + var_remote,
s_quantity = (case when s_quantity >= (var_ol_quantity + 10) then s_quantity else s_quantity + 91 end) - var_ol_quantity
WHERE s_i_id = var_ol_i_id
AND s_w_id = var_ol_supply_w_id
POLAR_RETURNING s_quantity, s_data,
(CASE in_d_id
WHEN 1 THEN s_dist_01
WHEN 2 THEN s_dist_02
WHEN 3 THEN s_dist_03
WHEN 4 THEN s_dist_04
WHEN 5 THEN s_dist_05
WHEN 6 THEN s_dist_06
WHEN 7 THEN s_dist_07
WHEN 8 THEN s_dist_08
WHEN 9 THEN s_dist_09
ELSE s_dist_10
END) AS dist
INTO var_s_quantity, var_s_data, var_ol_dist_info;

-- Set back err
SET err = 0;

-- get item info
SELECT i_price, i_name, (CASE WHEN locate("original", i_data) AND locate("original", var_s_data) then 'B' ELSE 'G' END) AS bg
INTO var_i_price, var_i_name, var_bg
FROM bmsql_item
WHERE i_id = var_ol_i_id;

```

```

-- calculate amount
SET var_ol_amount = var_ol_quantity * var_i_price;
SET out_total_amount = out_total_amount + var_ol_amount;

-- insert the order details
INSERT INTO bmsql_order_line (ol_o_id, ol_d_id, ol_w_id, ol_number, ol_i_id, ol_supply_w_id, ol_quantity, ol_amount, ol_dist_info)
VALUES (out_o_id, in_d_id, in_w_id, var_ol_number, var_ol_i_id, var_ol_supply_w_id, var_ol_quantity, var_ol_amount, var_ol_dist_info);

-- Set integrated variables
SET out_ol_supply_w_ids = concat_ws(',', out_ol_supply_w_ids, var_ol_supply_w_id);
SET out_i_ids = concat_ws(',', out_i_ids, var_ol_i_id);
SET out_ol_quantities = concat_ws(',', out_ol_quantities, var_ol_quantity);
SET out_item_names = concat_ws(',', out_item_names, var_i_name);
SET out_supply_quantities = concat_ws(',', out_supply_quantities, var_s_quantity);
SET out_brand_generic = concat_ws(',', out_brand_generic, var_bg);
SET out_prices = concat_ws(',', out_prices, var_i_price);
SET out_amounts = concat_ws(',', out_amounts, var_ol_amount);
SET var_ol_number = var_ol_number + 1;
END WHILE;

SET out_total_amount = out_total_amount * (1 + out_w_tax + out_d_tax) * (1 - out_c_discount);
IF err = 0 THEN
SELECT err, out_ol_supply_w_ids, out_i_ids, out_ol_quantities, out_item_names, out_supply_quantities,
      out_brand_generic, out_prices, out_amounts, out_total_amount, out_c_credit, out_c_discount, out_c_last,
      out_o_id, out_d_tax, out_o_entry_d, out_w_tax;
COMMIT;
END IF;
END $$

delimiter ;

-- order status
DROP PROCEDURE IF EXISTS bmsql_proc_order_status;
delimiter $$
CREATE PROCEDURE bmsql_proc_order_status(
  in in_w_id int,
  in in_d_id int,
  in in_c_id int,
  in in_by_lastname int,
  in in_c_last varchar(16))
label:BEGIN
-- The variables which will be returned
declare err int default 0;          -- -1 if there is an error, 0 otherwise
declare out_c_id int;              -- Customer id
declare out_c_balance decimal(12,2); -- Customer balance
declare out_c_first varchar(16);   -- Customer first name
declare out_c_middle char(2);      -- Customer middle name
declare out_c_last varchar(16);    -- Customer last name
declare out_o_id int;              -- Order id
declare out_o_entry_d datetime(3); -- Create time of order
declare out_o_carrier_id tinyint;  -- Carrier id

-- Integrated variables which will be returned
declare out_ol_i_ids varchar(512); -- Item ids
declare out_ol_supply_w_ids varchar(512); -- Warehouse ids of supplier
declare out_ol_quantities varchar(512); -- The quantity of each item
declare out_ol_amounts varchar(512); -- Total amount of each item
declare out_ol_delivery_dates varchar(512); -- Delivery date of each item

-- The variables used in procedure
declare var_ol_i_id int;           -- Item id
declare var_ol_supply_w_id int;    -- Warehouse id of supplier
declare var_ol_quantity tinyint;  -- Quantity of the item in the order
declare var_ol_amount decimal(6,2); -- Total amount (price * quantity)
declare var_done int default 0;    -- 1 if cursor is done
declare var_ol_delivery_d datetime(3); -- Date of delivery
declare var_name_cnt int;          -- Number of customer that matches the name
declare var_rowindex int;         -- Row index of the customer

```

```

declare exit HANDLER FOR NOT FOUND,SQLException
BEGIN
  SET err = -1;
  SELECT err, out_c_id, out_c_balance, out_c_first, out_c_middle, out_c_last, out_o_id,
    out_o_entry_d, out_o_carrier_id, out_ol_i_ids, out_ol_supply_w_ids, out_ol_quantities,
    out_ol_amounts, out_ol_delivery_dates;
  ROLLBACK;
END;
START TRANSACTION;
SET err = 0;
-- If in_by_lastname is not 0, get the customer by in_c_last
IF in_by_lastname != 0 THEN
  SELECT COUNT(c_id)
    INTO var_name_cnt
  FROM bmsql_customer
  WHERE c_w_id = in_w_id
    AND c_d_id = in_d_id
    AND c_last = in_c_last;

  -- If no record is found
  IF var_name_cnt = 0 THEN
    SET err = -1;
    SELECT err, out_c_id, out_c_balance, out_c_first, out_c_middle, out_c_last, out_o_id,
      out_o_entry_d, out_o_carrier_id, out_ol_i_ids, out_ol_supply_w_ids, out_ol_quantities,
      out_ol_amounts, out_ol_delivery_dates;
    ROLLBACK;
    LEAVE label;
  END IF;

  -- Find the customer
  SET var_rowindex = (var_name_cnt + 1)/2 - 1;

  -- Get customer id
  SELECT c_id
    INTO in_c_id
  FROM bmsql_customer
  WHERE c_w_id = in_w_id
    AND c_d_id = in_d_id
    AND c_last = in_c_last
  ORDER BY c_first
  LIMIT var_rowindex, 1;
END IF;

-- Get customer info
SET out_c_id = in_c_id;
SELECT c_balance, c_first, c_middle, c_last
  INTO out_c_balance, out_c_first, out_c_middle, out_c_last
FROM bmsql_customer
WHERE c_w_id = in_w_id
  AND c_d_id = in_d_id
  AND c_id = in_c_id;

-- find the most recent order for this customer
SELECT o_id, o_entry_d, COALESCE(o_carrier_id, 0)
  INTO out_o_id, out_o_entry_d, out_o_carrier_id
FROM bmsql_oorder
WHERE o_w_id = in_w_id
  AND o_d_id = in_d_id
  AND o_c_id = in_c_id
ORDER BY o_id DESC
LIMIT 1;

BEGIN
  DECLARE cur CURSOR FOR
    SELECT ol_i_id, ol_supply_w_id, ol_quantity, ol_amount, ol_delivery_d
      FROM bmsql_order_line
      WHERE ol_w_id = in_w_id
        AND ol_d_id = in_d_id
        AND ol_o_id = out_o_id;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET var_done = 1;
  OPEN cur;

```

```

WHILE var_done != 1 DO
  FETCH cur
  INTO var_ol_i_id, var_ol_supply_w_id, var_ol_quantity, var_ol_amount, var_ol_delivery_d;
  IF var_done != 1 THEN
    IF isnull(var_ol_delivery_d) THEN
      SET out_ol_i_ids = concat_ws(',', out_ol_i_ids, var_ol_i_id);
      SET out_ol_supply_w_ids = concat_ws(',', out_ol_supply_w_ids, var_ol_supply_w_id);
      SET out_ol_quantities = concat_ws(',', out_ol_quantities, var_ol_quantity);
      SET out_ol_amounts = concat_ws(',', out_ol_amounts, var_ol_amount);
      SET out_ol_delivery_dates = concat_ws(',', out_ol_delivery_dates, 'NULL');
    ELSE
      SET out_ol_i_ids = concat_ws(',', out_ol_i_ids, var_ol_i_id);
      SET out_ol_supply_w_ids = concat_ws(',', out_ol_supply_w_ids, var_ol_supply_w_id);
      SET out_ol_quantities = concat_ws(',', out_ol_quantities, var_ol_quantity);
      SET out_ol_amounts = concat_ws(',', out_ol_amounts, var_ol_amount);
      SET out_ol_delivery_dates = concat_ws(',', out_ol_delivery_dates, var_ol_delivery_d);
    END IF;
  END IF;
END WHILE;
CLOSE cur;
END;

SELECT err, out_c_id, out_c_balance, out_c_first, out_c_middle, out_c_last, out_o_id, out_o_entry_d,
  out_o_carrier_id, out_ol_i_ids, out_ol_supply_w_ids, out_ol_quantities, out_ol_amounts,
  out_ol_delivery_dates;
COMMIT;
END $$

delimiter ;

-- payment
DROP PROCEDURE IF EXISTS bmsql_proc_payment;
delimiter $$
-- out: err_num, date, warehouse street1, warehouse street2, warehouse city,
-- warehouse state, warehouse zip, district street1, district stree2,
-- district city, district state, district zip, customer first,
-- customer middle, customer last, customer street1, customer street2,
-- customer city, customer state, customer zip, customer phone,
-- customer since, customer credit, customer credit limitation,
-- customer discount, customer balance, customer data
CREATE PROCEDURE bmsql_proc_payment(
  in in_w_id int,           -- home warehouse number
  in in_d_id int,          -- the district number of the home warehouse
  in in_c_wid int,         -- customer warehouse number
  in in_c_did int,         -- customer district number
  in in_c_id int,          -- customer unique number when by id
  in in_amount decimal(12, 2), -- payment in_amount
  in in_by_lastname int,   -- 1 if the customer is selected based on customer last name, 0 otherwise
  in in_lastname char(17)) -- last name of the customer if in_by_lastname = 1, NULL otherwise
label:BEGIN

  -- The variables which will be returned
  declare err int;
  declare out_h_date char(24);
  declare out_w_street_1 varchar(20);
  declare out_w_street_2 varchar(20);
  declare out_w_city varchar(20);
  declare out_w_state char(2);
  declare out_w_zip char(9);
  declare out_d_street_1 varchar(20);
  declare out_d_street_2 varchar(20);
  declare out_d_city varchar(20);
  declare out_d_state char(2);
  declare out_d_zip char(9);
  declare out_c_id int;
  declare out_c_first varchar(16);
  declare out_c_middle char(2);
  declare out_c_last char(17);
  declare out_c_street_1 varchar(20);
  declare out_c_street_2 varchar(20);
  declare out_c_city varchar(20);

```

```

declare out_c_state char(2);
declare out_c_zip char(9);
declare out_c_phone char(16);
declare out_c_since char(81);
declare out_c_credit char(2);
declare out_c_creditlim bigint;
declare out_c_discount decimal(4,2);
declare out_c_balance decimal(12,2);
declare out_c_data text(201);

-- The variables used in procedure
declare var_w_name char(11);    -- warehouse name
declare var_d_name char(11);    -- district name
declare var_c_cnt int;         -- number of rows that match the lastname
declare var_c_id_vals text;    -- customer id

declare exit HANDLER FOR NOT FOUND,SQLEXCEPTION
BEGIN
    SET err = -1;
    SELECT err, out_h_date, out_w_street_1, out_w_street_2, out_w_city, out_w_state, out_w_zip, out_d_street_1, out_d_street_2, out_d_city,
out_d_state, out_d_zip,
        out_c_first, out_c_middle, out_c_last, out_c_street_1, out_c_street_2, out_c_city, out_c_state, out_c_zip, out_c_phone, out_c_since,
out_c_credit,
        out_c_creditlim, out_c_discount, out_c_balance, out_c_data;
    ROLLBACK;
END;

START TRANSACTION;
SET err = 0;
SELECT CURRENT_TIMESTAMP(3) INTO out_h_date;
SET out_c_id=in_c_id;

IF in_by_lastname != 0 THEN
    -- When the customer is selected based on customer last name, get the sorted set.
    -- When no customer is found, will get NOT FOUND condition.
    SELECT group_concat(c_id ORDER BY c_first), count(c_id)
        INTO var_c_id_vals, var_c_cnt
    FROM bmsql_customer
    WHERE c_w_id = in_c_wid AND c_d_id = in_c_did AND c_last = in_lastname;

    -- Get the customer id from position(cnt/2 rounded up to the next integer) in the sorted set.
    SELECT substring_index(substring_index(var_c_id_vals, ',', ceil(var_c_cnt/2)), ',', -1)
    INTO out_c_id;
END IF;

-- Update customer balance and payment, and get the customer information.
UPDATE bmsql_customer
SET c_balance= c_balance - in_amount, c_ytd_payment = c_ytd_payment + in_amount, c_payment_cnt = c_payment_cnt + 1
WHERE c_w_id = in_c_wid AND c_d_id = in_c_did AND c_id = out_c_id
POLAR_RETURNING c_first, c_middle, c_last, c_street_1, c_street_2, c_city, c_state, c_zip, c_phone, c_since, c_credit, c_credit_lim, c_discount,
c_balance
    INTO out_c_first, out_c_middle, out_c_last, out_c_street_1, out_c_street_2, out_c_city, out_c_state, out_c_zip, out_c_phone, out_c_since,
out_c_credit, out_c_creditlim, out_c_discount, out_c_balance;

IF out_c_credit="BC" THEN
    -- Update customer C_DATA when C_CREDIT is equal to 'BC'.
    UPDATE bmsql_customer
    SET c_data = substr(concat(out_c_id, ',', in_c_did, ',', in_c_wid, ',', in_d_id, ',', in_w_id, ',', in_amount, ' | ', c_data), 1, 500)
    WHERE c_w_id = in_c_wid AND c_d_id = in_c_did AND c_id = out_c_id
POLAR_RETURNING substr(c_data, 1, 200)
    INTO out_c_data;
ELSE
    -- When C_CREDIT is equal to 'GC'.
    SET out_c_data = "";
END IF;

-- Update district balance, and get district information.
UPDATE bmsql_district
SET d_ytd = d_ytd + in_amount
WHERE d_w_id = in_w_id AND d_id = in_d_id
POLAR_RETURNING d_name, d_street_1, d_street_2, d_city, d_state, d_zip

```

```

    INTO var_d_name, out_d_street_1, out_d_street_2, out_d_city, out_d_state, out_d_zip;

-- Update warehouse balance, and get warehouse information.
UPDATE bmsql_warehouse
SET w_ytd = w_ytd + in_amount
WHERE w_id = in_w_id
POLAR_RETURNING w_name, w_street_1, w_street_2, w_city, w_state, w_zip
    INTO var_w_name, out_w_street_1, out_w_street_2, out_w_city, out_w_state, out_w_zip;

-- Insert new row into history table.
INSERT INTO bmsql_history (h_c_id, h_c_d_id, h_c_w_id, h_d_id, h_w_id, h_date, h_amount, h_data)
VALUES (out_c_id, in_c_did, in_c_wid, in_d_id, in_w_id, out_h_date, in_amount, concat(var_w_name, ' ', var_d_name));

IF err = 0 THEN
    SELECT err, out_h_date, out_w_street_1, out_w_street_2, out_w_city, out_w_state, out_w_zip, out_d_street_1, out_d_street_2, out_d_city,
    out_d_state, out_d_zip,
           out_c_first, out_c_middle, out_c_last, out_c_street_1, out_c_street_2, out_c_city, out_c_state, out_c_zip, out_c_phone, out_c_since,
    out_c_credit,
           out_c_creditlim, out_c_discount, out_c_balance, out_c_data;
COMMIT;
ELSE
    SET err = -1;
ROLLBACK;
END IF;
COMMIT;
END $$

delimiter ;

-- delivery
DROP PROCEDURE IF EXISTS bmsql_proc_delivery_bg;
delimiter $$
CREATE PROCEDURE bmsql_proc_delivery_bg(
    in in_w_id int,
    in in_o_carrier_id int)
BEGIN
    -- The variables which will be returned
    declare out_delivered_o_ids varchar(128);    -- delivery id of each order
    declare errinfo varchar(128);              -- err info

    -- The variables used in procedure
    declare var_d_id int default 1;            -- District id
    declare var_o_id int;                      -- Order id
    declare var_c_id int;                      -- Customer id
    declare var_ol_delivery_d datetime(3);    -- Delivery date
    declare var_total_ol_amount decimal(12,2); -- Sum of amount

    SELECT CURRENT_TIMESTAMP(3) INTO var_ol_delivery_d;
    START TRANSACTION;

    myloop:
    WHILE var_d_id <= 10 DO
        -- The 'polar_returning' IS to see which ORDER IS deleted FROM bmsql_new_order.
        DELETE FROM bmsql_new_order
        WHERE no_w_id = in_w_id
            AND no_d_id = var_d_id
        ORDER BY no_o_id ASC
        LIMIT 1
        POLAR_RETURNING no_o_id
            INTO var_o_id;

        IF found_rows() > 0 THEN
            UPDATE bmsql_oorder
            SET o_carrier_id = in_o_carrier_id
            WHERE o_w_id = in_w_id
                AND o_d_id = var_d_id
                AND o_id = var_o_id
            polar_returning o_c_id
                INTO var_c_id;

            IF ROW_COUNT() != 1 THEN

```



```

        SET errinfo = concat("update bmsql_oorder cannot find, for no_d_id:",var_d_id);
        LEAVE myloop;
    END IF;

    UPDATE bmsql_order_line
        SET ol_delivery_d = var_ol_delivery_d
    WHERE ol_o_id = var_o_id AND ol_d_id = var_d_id AND ol_w_id = in_w_id;

    IF ROW_COUNT( ) < 1 THEN
        SET errinfo = concat("update bmsql_order_line cannot find, for no_d_id:",var_d_id);
        LEAVE myloop;
    END IF;

    SELECT SUM(ol_amount)
        INTO var_total_ol_amount
    FROM bmsql_order_line
    WHERE ol_o_id = var_o_id
        AND ol_d_id = var_d_id
        AND ol_w_id = in_w_id;

    IF ROW_COUNT( ) != 1 THEN
        SET errinfo = concat("sum(var_total_ol_amount) cannot find, for no_d_id:",var_d_id);
        LEAVE myloop;
    END IF;

    UPDATE bmsql_customer
        SET c_balance = c_balance + var_total_ol_amount , c_delivery_cnt = c_delivery_cnt + 1
    WHERE c_id = var_c_id
        AND c_d_id = var_d_id
        AND c_w_id = in_w_id;

    IF ROW_COUNT( ) != 1 THEN
        SET errinfo = concat("update bmsql_customer cannot find, for no_d_id:",var_d_id);
        LEAVE myloop;
    END IF;

    SET out_delivered_o_ids = concat_ws(',', out_delivered_o_ids, var_o_id);
END IF;

SET var_d_id = var_d_id + 1;

END WHILE myloop;

SELECT out_delivered_o_ids, errinfo;
IF ISNULL(errinfo) THEN
    COMMIT;
ELSE
    ROLLBACK;
END IF;
END $$

delimiter ;

-- stock level
DROP PROCEDURE IF EXISTS bmsql_proc_stock_level;
delimiter $$
CREATE PROCEDURE bmsql_proc_stock_level(
in in_w_id int,
    in in_d_id int,
    in in_level int)
BEGIN
declare out_low_stock int; -- Count of result rows

START TRANSACTION;

SELECT COUNT(DISTINCT (s_i_id))
    INTO out_low_stock
FROM bmsql_order_line, bmsql_stock, bmsql_district
WHERE d_w_id = ol_w_id
    AND ol_w_id = s_w_id
AND d_w_id = in_w_id

```

```

AND d_id = ol_d_id
AND d_id = in_d_id
AND ol_i_id = s_i_id
AND s_quantity < in_level
AND ol_o_id BETWEEN (d_next_o_id - 20) AND (d_next_o_id - 1);

```

```

SELECT out_low_stock;
COMMIT;
END $$

```

delimiter ;

Appendix C: Configuration Options

The settings that have been changed from the defaults found in the actual software products are provided as follows:

C.1 F62M1 OS configuration

#File system Info

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
devtmpfs	395803736	0	395803736	0%	/dev
tmpfs	395819908	1640	395818268	1%	/dev/shm
tmpfs	395819908	2200	395817708	1%	/run
tmpfs	395819908	0	395819908	0%	/sys/fs/cgroup
/dev/sda3	98559580	15510972	77999060	17%	/
/dev/sda4	130105436	2459100	120994280	2%	/home
/dev/sda2	999320	152316	778192	17%	/boot

#OS version

```

NAME="Alibaba Group Enterprise Linux Server"
VERSION="7.2 (Paladin)"
ID="alios"
ID_LIKE="fedora anolis"
VERSION_ID="7.2"
PRETTY_NAME="Alibaba Group Enterprise Linux Server 7.2 (Paladin)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:alibaba:enterprise_linux:7.2:GA:server"
HOME_URL="https://os.alibaba-inc.com/"
BUG_REPORT_URL="https://os.alibaba-inc.com/"

```

```

ALIBABA_BUGZILLA_PRODUCT="Alibaba Group Enterprise Linux 7"
ALIBABA_BUGZILLA_PRODUCT_VERSION=7.2
ALIBABA_SUPPORT_PRODUCT="Alibaba Group Enterprise Linux"
ALIBABA_SUPPORT_PRODUCT_VERSION=7.2

```

#kernel version

```
Linux u89d11262.cloud.eo166 4.19.91-008.ali4000.alios7.x86_64 #1 SMP Fri Sep 4 17:33:26 CST 2020 x86_64 x86_64 x86_64 GNU/Linux
```

C.2 Nginx configuration

```

worker_processes {RTE_PROCESS};
worker_cpu_affinity auto;

```

```
error_log logs/error.log warn;
```

```

events {
    worker_connections 600000;
}

```

```

http {
    include mime.types;
    default_type application/octet-stream;

    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
'$status $body_bytes_sent $request_body';

    access_log off;
}

```

```

sendfile on;

keepalive_timeout 32400;
keepalive_time 32400;
keepalive_requests 10000000;
lua_shared_dict BROKEN_BACKENDS 1m;
lua_check_client_abort on;

init_worker_by_lua_block {
    DB_ADDR_LIST = {
        {RTE_PROXY_LIST}
    }
    PGCONN_TIMEOUT = 60 * 1000
    PGCONN_KEEPALIVE_TIMEOUT = 300 * 1000
    PGCONN_POOL_SIZE = 300
    TPCC_BACKEND_SIZE = {RTE_BACKEND_SIZE}

    package.path = package.path .. ";/nginx/lua/?lua;"
    local dp = require("delivery_processor")
    local num_workers = 16
    local common = require("common")
    local start = function()
        dp.start(num_workers)
        common.start_monitor()
    end

    ngx.timer.at(1, start)
}

server {
    {RTE_LISTEN_LIST}

    server_name 0.0.0.0;
    root html;
    index index.html;
    default_type text/html;
    autoindex on;
    lua_socket_pool_size 16;

    proxy_read_timeout 600;

    location /login {
        rewrite ^ /menu.html?$args permanent;
    }

    location ~ /tpcc/(.+) {
        content_by_lua_file lua/$1.lua;
    }

    location /nginx_status {
        stub_status on;
        access_log off;
        allow 127.0.0.1;
        deny all;
    }
}
}

```

C.3 Data Node database configuration

```

[mysqld]
temptable_max_mmap = 107374182400
innodb_csn = ON
innodb_csn_distributed = ON
wide_primary_fast_lookup = ON
eq_range_use_index_stats = ON
innodb_primary_sync_slave = 2
innodb_buffer_pool_in_core_file = OFF
innodb_lock_wait_timeout = 30
innodb_csn_log_size = 536870912

```

innodb_lf_hash_shard_count = 32
skip_name_resolve = ON
max_connections=20000
back_log=65535
default_password_lifetime=0
ssl=0
max_prepared_stmt_count=512000
skip_log_bin=1
character_set_server=latin1
collation_server=latin1_swedish_ci
transaction_isolation=REPEATABLE-READ
innodb_file_per_table
innodb_log_file_size=1024M
innodb_log_files_in_group=3
innodb_open_files=4000
innodb_log_buffer_size=64M
innodb_thread_concurrency=0
innodb_max_dirty_pages_pct=90
innodb_max_dirty_pages_pct_lwm=10
join_buffer_size=32K
sort_buffer_size=32K
innodb_use_native_aio=1
innodb_stats_persistent=1
innodb_spin_wait_delay=6
innodb_max_purge_lag_delay=300000
innodb_max_purge_lag=0
innodb_checksum_algorithm=crc32
innodb_lru_scan_depth=9000
innodb_change_buffering=none
innodb_read_only=0
innodb_undo_log_truncate=off
innodb_adaptive_flushing=1
innodb_flush_neighbors=0
innodb_read_io_threads=16
innodb_purge_threads=32
innodb_adaptive_hash_index=0
innodb_monitor_enable='%'
innodb_random_read_ahead = ON
innodb_use_native_aio=0
innodb_flush_method=O_DIRECT_NO_FSYNC
innodb_flush_log_at_trx_commit=1
table_open_cache_instances=16
innodb_io_capacity=10000
innodb_io_capacity_max=20000
innodb_undo_tablespaces=127
performance_point_enabled=off
innodb_flush_sync=OFF
innodb_page_size=4k
innodb_doublewrite=0
default_authentication_plugin=mysql_native_password
performance_schema=OFF

Appendix D: Price References

D.1 Alibaba Cloud PolarDB Limitless with 2340 polar.mysql.mmx10.6xlarge and 31.5TB storage each

主可用区 可用区I 12 可用区K 15
可用区是地域中的一个独立物理区域。不同可用区之间没有实质性区别。

网络类型 专有网络
可用私有IP数量 247 个
如需创建新的专有网络，您可点击[创建](#)。如需创建新的交换机，您可点击[创建](#)。

加入白名单 开启
将VPC网段加入到PolarDB集群白名单中，使得同一VPC下的ECS实例可以访问PolarDB集群

高可用模式 **双AZ (开启存储热备集群)** 单AZ (关闭存储热备集群)
使用须知 已开启存储热备的集群不支持关闭本功能。在同地域下同时部署主集群和存储热备集群(两份存储)，更高 SLA 可靠性保障。地域支持多可用区情况下，跨可用区部署热备集群，具备可用区容灾能力。 [详细参考](#)。

备可用区 自动分配 可用区K 15

规格和代理

筛选 CPU 内存 规格代码

规格族	规格代码	CPU(核)	内存(GB)	最大存储空间	PSL4 最大 IOPS	PSL5 最大 IOPS	参考价格(单节点)
独享规格	polar.mysql.mmx4.2xlarge	16	64 GB	100TB	50000	192000	¥ 4,000.00 /月
独享规格	polar.mysql.mmx8.2xlarge	16	128 GB	100TB	50000	192000	¥ 6,400.00 /月
独享规格	polar.mysql.mmx4.4xlarge	32	128 GB	100TB	80000	288000	¥ 8,000.00 /月
独享规格	polar.mysql.mmx8.4xlarge	32	256 GB	300TB	80000	288000	¥ 12,800.00 /月
独享规格	polar.mysql.mmx10.6xlarge	48	512 GB	100TB	150000	300000	¥ 13,200.00 /月
独享规格	polar.mysql.mmx8.8xlarge	64	512 GB	500TB	100000	288000	¥ 28,160.00 /月
独享规格	polar.mysql.mmx8.12xlarge	88	710 GB	500TB	150000	384000	¥ 35,200.00 /月

当前选择规格 polar.mysql.mmx10.6xlarge (48 核 512 GB, 独享规格)
规格详情和更多规格详见 [产品规格](#)。定价详见 [参考文档](#)。

节点个数 2340 个读写节点 0 个全局只读节点

存储

存储类型 PSL5 PSL4 **PSL3**

存储计费类型 按容量计费 (按量计费) 按空间计费 (包年包月)

存储空间 31540 GB
最大IOPS: --, 参考价格: ¥2,125,543,680.00

购买数量 1

购买时长 3年 **5折**

自动续费 启动自动续费

配置概要

计费类型 包年包月
地域 华北2 (北京) 可用区I
创建方式 创建主集群
数据库引擎 MySQL 8.0.2
产品版本 企业版
系列 多主集群 (Limitless)
子系列 独享规格
CPU架构 X86
只读节点个数 0
读写节点个数 2340
主可用区 可用区I
VPC网络 fengyi-test-vpc-1115 | 10.0.0.0/8
VPC交换机 fengyi-test-vsw-i-1115 | 10.10.0.0/24
高可用模式 **双AZ (开启存储热备集群)**
备可用区 自动分配
节点规格 独享规格 / polar.mysql.mmx10.6xlarge(48vC 512 GB)
存储类型 PSL3
存储计费类型 按空间计费 (包年包月)
存储空间 31540
资源组 default resource group
参数模板 -
时区 UTC +08:00 (默认)
表名大小写 不区分大小写 (默认)
是否开启Binlog 关闭
删除 (释放) 集群时 保留最后一个备份 (释放前自动备份) (默认)

应付费用: **¥1,618,755,840.00**
3年付立享5折优惠 省¥1,618,755,840.00 [查看详情](#)

点击立即购买即表示您已知悉并同意 [产品服务协议](#)、[服务等级协议](#) 以及本页面中您勾选过的产品专属条款 (若有)

您可以点此查看订单对应的发票信息。请在 [管理控制台-费用中心-发票管理](#) 中查看。也可以使用 [价格计算器](#) 进行预估整体费用。

[加入购物车](#) [立即购买](#)

D.2 14 英寸 MacBook Pro



14 英寸 MacBook Pro - 深空黑色 1 **RMB 15,999**

[显示产品详情](#) [移除](#)

添加 AppleCare+ 服务计划 (适用于配备 M4 芯片的 14 英寸 MacBook Pro) - RMB 2,299 [添加](#)

获得长达 3 年的技术支持和意外损坏维修服务。
[进一步了解](#)

📦 2-4 个工作日发货。 [了解你何时能收到这件商品。选择地区](#)

📍 目前暂不提供 Apple Store 零售店取货服务 [查找零售店](#)

小计	RMB 15,999
运费	免费
总计	RMB 15,999
每月分期付款	RMB 667/月(0% 费率 24 个月)

[可享免息分期付款](#)

D.3 OpenResty Support Price



报价单

客户名称	阿里云计算有限公司	供货单位	鸥锐软件开发(天津)有限公司 为OpenResty Inc 中国全资子公司
地址	浙江省杭州市西湖区三墩镇灯彩街1008号云谷园1-3-C08	地址	天津市武清区京津电子商务产业园宏旺道2号13号楼279室
电话	19257850424	电话	13366586563
传真	19257850424	传真	13366586563
联系人	周攀峰	联系人	刘继敏
手机	19257850424	手机	13366586563
邮箱	panfeng.zhou@alibaba-inc.com	邮箱	admin@openresty.com
报价日期	2025年1月26日	项目	软件技术支持

感谢您的垂询。我们很高兴给您提供如下报价，价格明细表如下：

条目	服务项目	优惠后单价 (CNY)	服务器数量	优惠后总价 (CNY)
技术支持	OpenResty Professional Edition Support Service for 3 years 7×24, 目标响应时间: 4小时 OpenResty 1.25.3.1 Nginx 1.25.3.1 OpenResty Inc OpenResty 和Lua私有库模块在合同期内的使用权 112个 OpenResty XRay Agent 云版动态追踪软件的使用权 复杂问题的技术支持依赖 OpenResty XRay 动态追踪软件 CPU体系结构: x86_64	14,000	560	7,840,000
总计	¥7,840,000 (大写: 柒佰捌拾肆万元)			

备注

1. 以上报价30天内有效，超过30天请重新询价。
2. 此报价的6%增值税作为优惠予以减免。
3. 此报价原价16,000元/服务器，原价总计：¥8,960,000; 总计优惠：¥1,120,000元。
4. 付款方式，合同签署后30天内100%预付。
5. 此报价仅限客户需求数量报价，如需增加服务器数量可以按此报价，但不支持减少服务器数量。如果对上述条款有任何问题，请随时向我们咨询。

报价单位：鸥锐软件开发(天津)有限公司

报价日期：2025年1月26日

