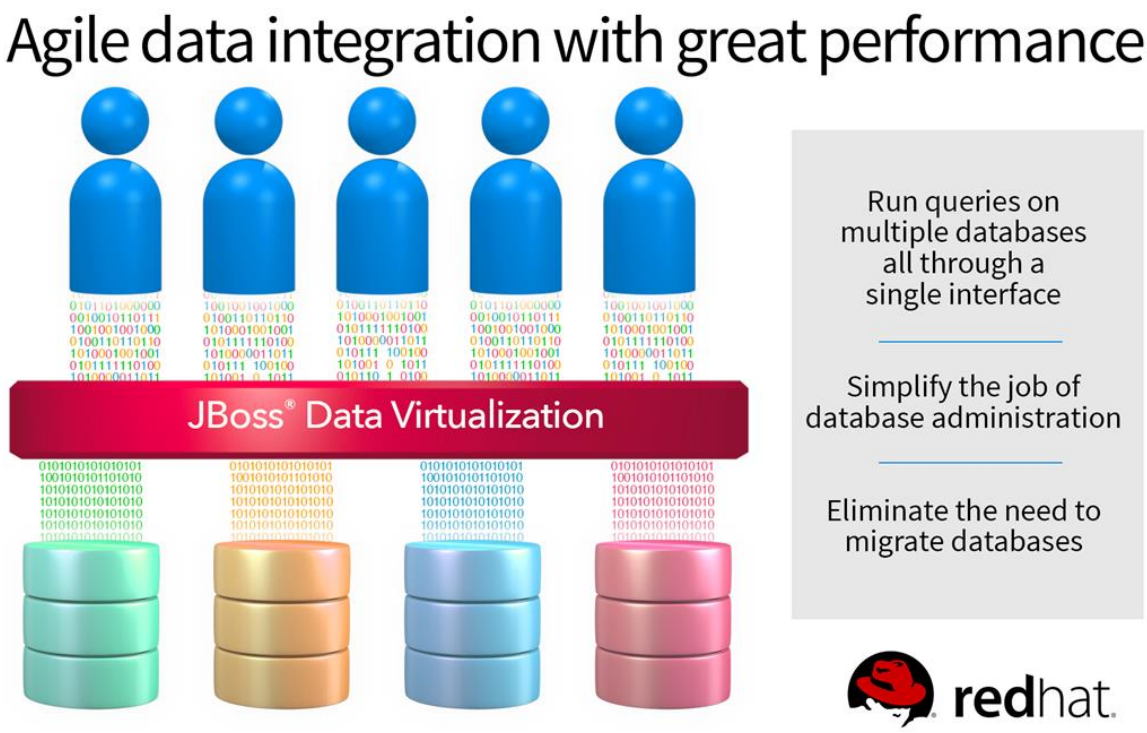
QUERYING VIRTUALIZED DATABASES WITH RED HAT JBOSS DATA VIRTUALIZATION



Enterprises today struggle with data challenges that include scattered data across multiple heterogeneous sources, data that is incomplete and hard to understand, and new or changing data sources that must be integrated or updated quickly. Data virtualization addresses these challenges by providing standards-based connectivity that can hide complexities of underlying data sources. In addition, data virtualization can allow your business to access data from multiple sources without migrating or copying any data, potentially improving accuracy and agility while reducing costs. Data virtualization can enable enterprises to access data through tools already in house, making the data consumable by any standards-based application.

When introducing data virtualization into your architecture, questions may arise. In what scenarios is this technology viable? How well can it perform? Does this technology offer the flexibility to work with existing tools and diverse data sources as well as the agility to incorporate new data sources?

In the Principled Technologies datacenter, we wanted to understand the use and role of data virtualization as it can apply to enterprise databases. We looked at what kinds of use cases for which an organization could effectively use Red Hat JBoss Data Virtualization (JDV) and then measured its querying performance in a number of environments and situations. We found that in our use cases, including querying a single data source with JDV, a federated set of data sources via JDV, in both transactional and analytical scenarios, JDV performed favorably and scaled, in some cases performing better than querying the native data sources.

AUGUST 2015
A PRINCIPLED TECHNOLOGIES TEST REPORT
Commissioned by Red Hat Inc.

POTENTIAL JBOSS DATA VIRTUALIZATION USE CASES FOR YOUR BUSINESS

Red Hat JDV can add value to your business. We chose four potential use cases that cross a range of workloads common in large organizations and use a varying number of data sources. We used four relational database management systems (RDBMS), which we refer to as Databases A, B, C, and D. Our testing measured query latency and query throughput to demonstrate the advantage of using JDV. In addition, we closely monitored resource consumption stability of the systems. For more information on the RDBMS we used for this testing, see Appendix A.

Use case 1 – Querying one database instance: Directly vs. pushdown using JDV

In this use case, a user executes a set of SQL queries to the relational database management system (RDBMS) using Java database connectivity (JDBC) technology. Our testing compared querying directly from a client machine to the RDBMS using the vendor-specific JDBC and querying against JDV using the JDV JDBC driver (see Figure 1). For this use case, we used Database A. For more information on JDV and the queries we used, see Appendix A and Appendix B.

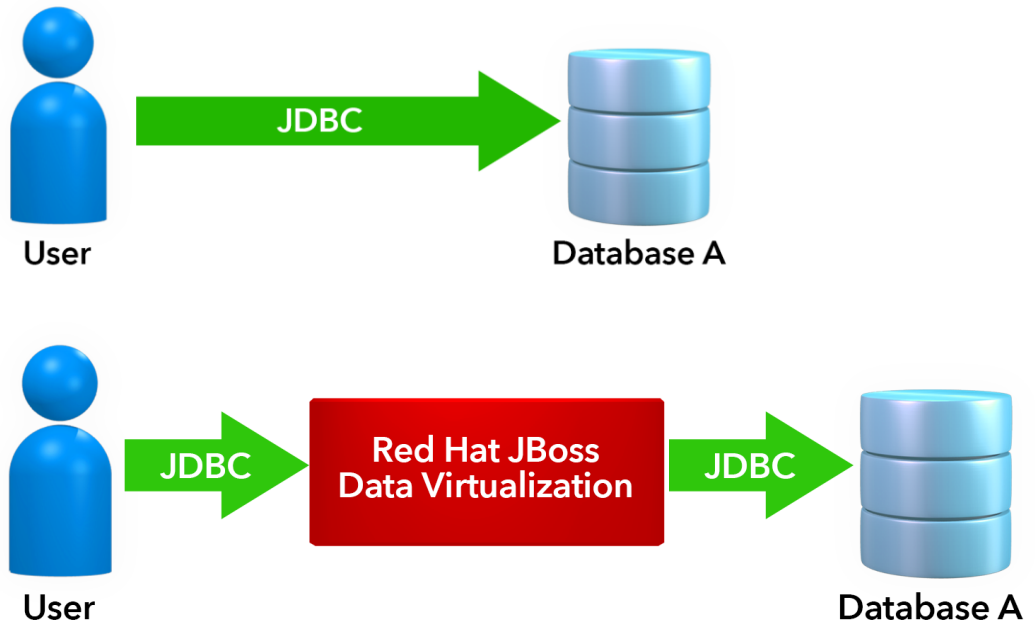


Figure 1: JDV sits between the user and the data sources, and uses SQL queries and JDBC to interrogate the data source.

Use case 2 – Querying a virtual database for an analytical workload

In this use case, a user executes the same set of queries against JDV; however, the data comes from four RDBMS (Databases A, B, C, and D) rather than one (see Figure 2). A JDV "virtual database" presents a unified, integrated database that represents the complete data set.

This use case simulates a typical Business Intelligence (BI) analytical use case for enterprises, such as calculating sales trends for a particular product across all customers from multiple databases over the course of a month. These queries can take a long time because they read large amounts of data from source databases and then perform data integration in the JDV layer.

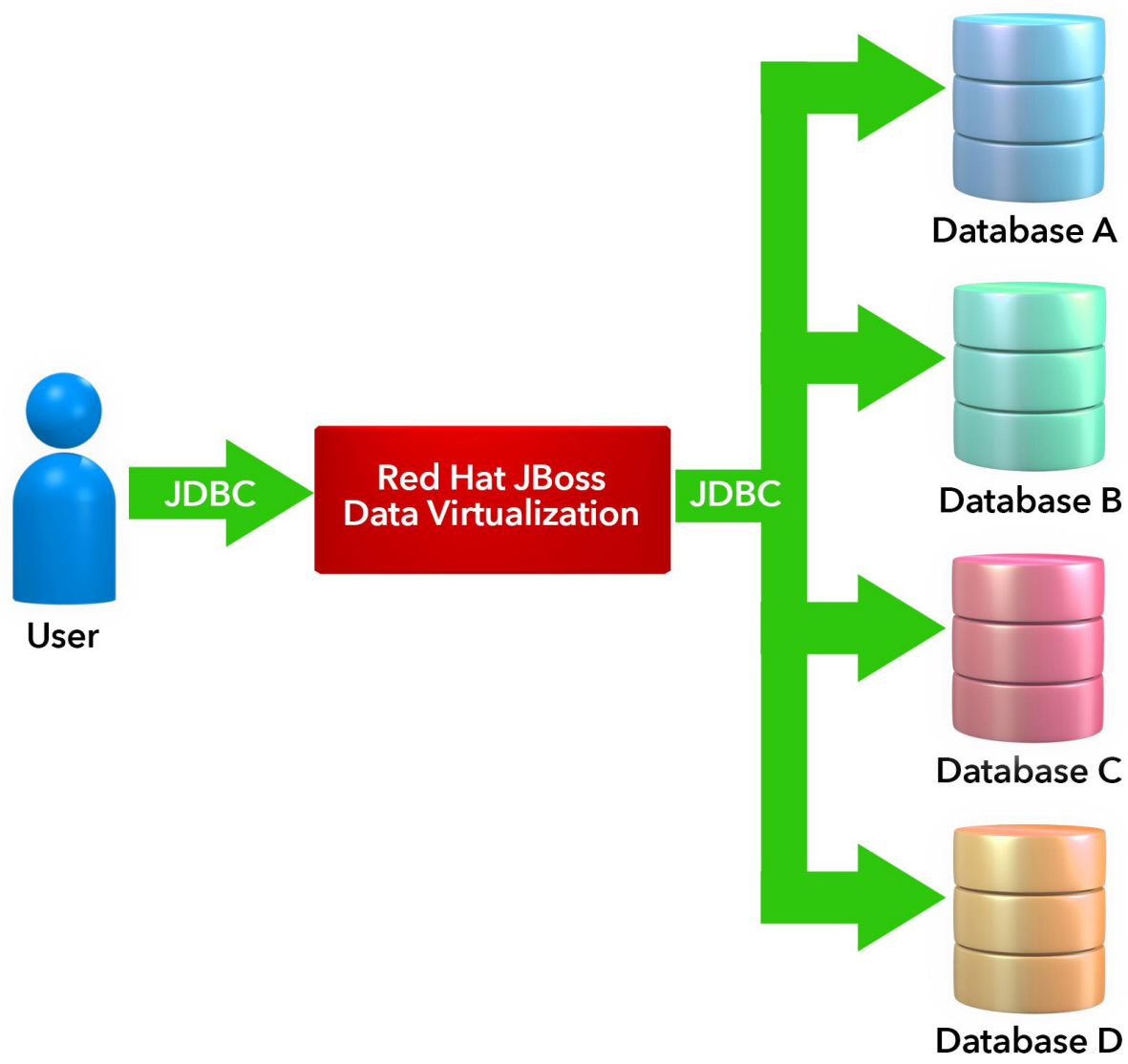


Figure 2: Querying against JDV for data from four RDBMS.

Use case 3 – Scaling the queries of an analytical workload

Similar to use case 2, this use case also features an analytical load but varies the number of concurrent users to simulate small, medium, and large teams of BI analysts querying the data sources simultaneously for analytic queries that return large results (see Figure 3). This scenario observes the stability and performance of the virtualized-database system.

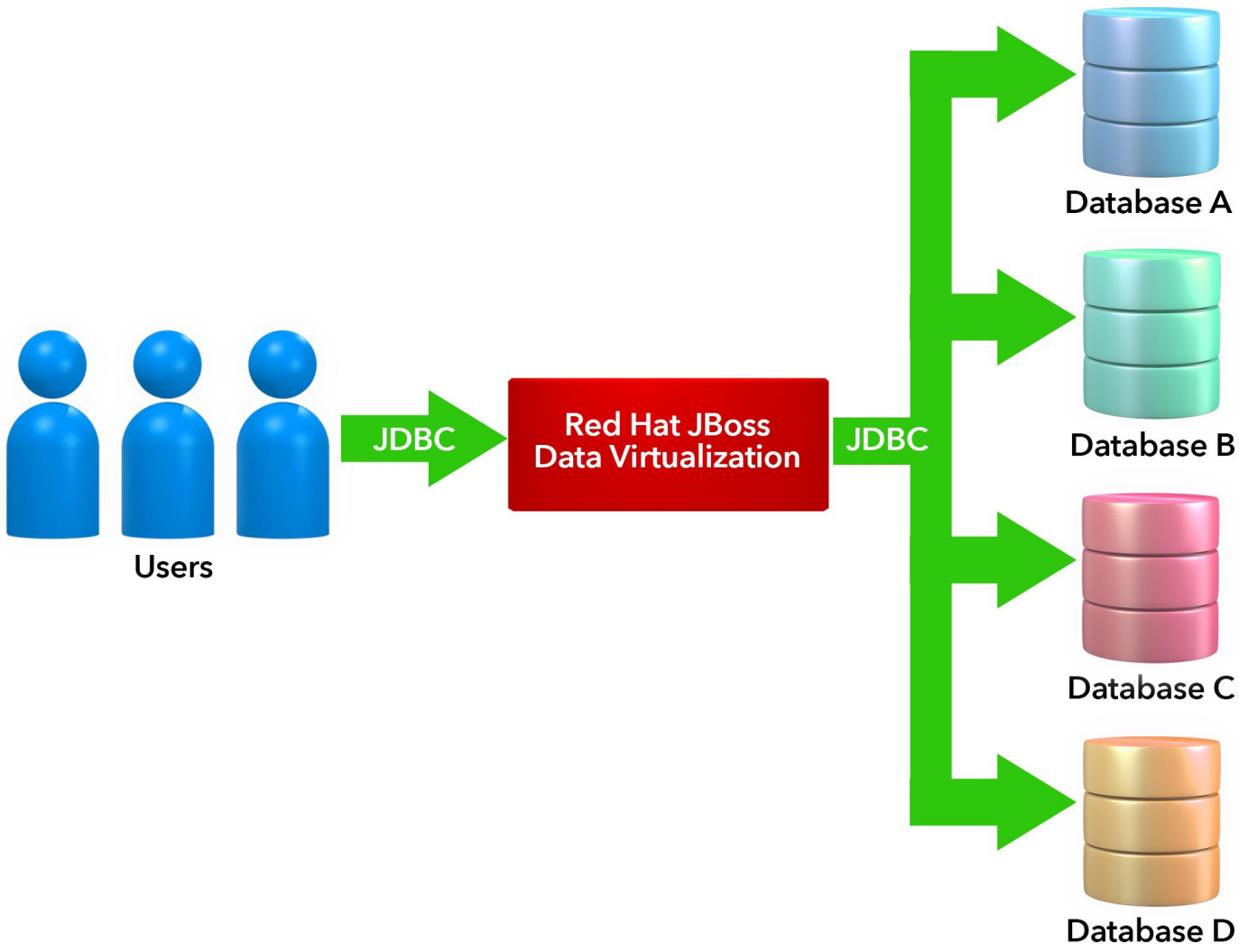


Figure 3: Multi-user querying against JDV for data from four RDBMS.

Use case 4 – Querying a virtual database for an online transaction processing (OLTP) workload

This scenario switches from BI use cases to transactional processing for many simultaneous users. The data is distributed across databases exactly as in use case 3, and JDV presents the same "virtual database" to the users. However, the query executed represents an online transaction processing (OLTP) query.

OLTP queries are typically executed in operational datastores, such as an online order system or customer database. Queries in this scenario need to have faster response times and to return smaller amounts of data compared to use case 3. An example is looking up a single customer or the customer's order history. The scenario runs a single query that returns a small result with high levels of concurrent users.

ADVANTAGES OF USING VIRTUAL DATABASES WITH RED HAT JDV

Information about a single entity, such as a customer or item, can exist in many different parts of your business. For example, manufacturing companies may have separate databases for manufacturing, sales, research and development, and shipping. When your business needs to integrate data from these multiple sources, the process can be cumbersome—having different kinds of data can complicate data analysis and database development efforts.

Merging data sets may sound like a good solution to getting all your data in one place for querying, but this can be expensive, require costly application changes and many hours from IT, and increase the potential for human error. In addition, DBMS semantics and security capabilities vary, which further complicates consolidation efforts, and programming (for queries, etc.) means additional time and costs. Ultimately, if your databases are working well, you may not want to disrupt your database infrastructure.

Using data integration software such as JBoss Data Virtualization (JDV) with multiple data sources can offer the benefits of data consistency and access while helping your business avoid the potential issues involved with merging datasets. With JDV, your business can use a common interface to query data residing in multiple data sources without having to integrate datasets. Whatever the reason for having multiple data sources, unifying them with JDV can make accessing data easier, with minimal interruption for your users, while potentially easing the labor burden on your IT, database administrators, and database developers.[1]

[1] For more information on Red Hat JDV, see Appendix A.

Test environment

Hardware

Figure 4 shows our specifications and roles for the servers we used.

Model	Processors	Memory (GB)	Storage	Functional role
Lenovo ThinkServer RD550	Intel Xeon E5-2699 v3 (2)	128	HDDs; 1.1 TB total	JBoss Data Virtualization 6.1.0 host
Lenovo ThinkServer RD550	Intel Xeon E5-2699 v3 (2)	128	SSDs, HDDs; 1.8 TB total	Database A server, hosting 1 TB data source
Lenovo ThinkServer RD550	Intel Xeon E5-2698 v3 (2)	128	SSDs, HDDs; 1.8 TB total	Database B server, hosting 1 TB data source
Lenovo ThinkServer RD540	Intel Xeon E5-2690 v2 (2)	128	HDDs; 1.6 TB total	Database C server, hosting 1 TB data source
Lenovo ThinkServer RD540	Intel Xeon E5-2690 v2 (2)	128	HDDs; 1.6 TB total	Database D server, hosting 1 TB data source

Figure 4: Our test servers and functional roles. See Appendix A for complete list.

Operating systems

- Red Hat Enterprise Linux 7 on the JDV, database, and test-harness hosts
- Microsoft Windows 2012 R2

Software versions

- JBoss Data Virtualization version 6.1
- Apache JMeter v2.13
- See Appendix A for RDBMS details

Network

- All machines are configured with 10GbE LAN.

Source database(s) setup

The TPC Benchmark H (TPC-H) is a decision-support benchmark that consists of a suite of business oriented ad-hoc queries and concurrent data modifications. We used TPC-H-like schema and data for performance testing and loaded each database (Databases A, B, C, and D) with 1 TB of TPC-H-like data. The 1 TB of data represents about 150 million customers, with over 600 million order records, and 6 billion order line items. Our test queries were also TPC-H-like and labeled A through H.[2]

Physical architecture

The JDV environment consisted of one JDV server in front of four distinct database servers. For the final federated-data scenario, we completed queries with a mix of numbers of concurrent users. Figure 5 shows the flow of queries in the physical architecture of our test solution.

[2] For detailed information on the test queries, see Appendix B.

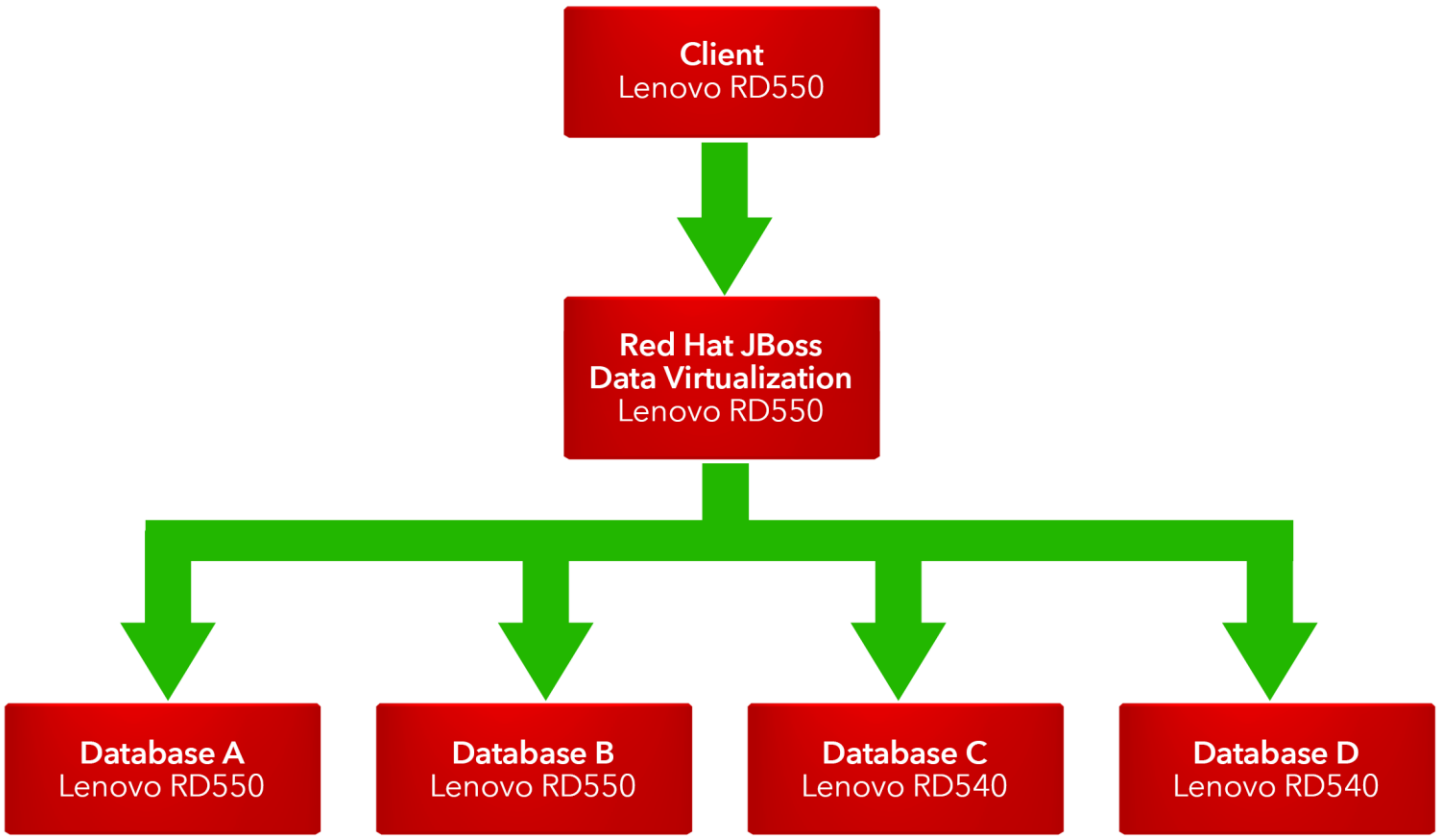


Figure 5: Architecture of our test solution.

What we found

Use Case 1 - Direct to Database A vs. pushdown to Database A using JDV

To establish a baseline set of results to compare against the results from using JDV pushdown, we configured the vendor-specific Database A JDBC driver directly to Database A and executed the queries with Apache JMeter.

Then, instead of directly accessing Database A, we set up the JDV server between the database and the client. This allowed the client to interact with the JDV system, which then interacted with the database. We configured JDV to perform the data federation, where the results returned to the client from JDV in the same form as when they returned from Database A. Figure 6 shows our baseline and JDV pushdown response times for each query. For the number of rows returned per query, see Appendix B.

Query	Single-users response times (seconds)	
	Direct query to Database A	JDV pushdown to one database (Database A)
A	0.002	0.016
B	58.758	56.012
C	0.285	0.165
D	0.057	0.074
E	4.156	4.803
F	79.504	78.938
Total	**142.762**	**140.008**

Figure 6: Response times when directly querying Database A (baseline) and when querying it with JDV pushdown.

Comments on use case 1

We configured JDV as a simplistic virtualization layer between the user and the Database A database. Testing in this way can demonstrate the additional overhead introduced by JDV in terms of latency.

Total response time after inserting JDV between the client and the Database A database was 2 seconds less than direct queries through the vendor-specific JDBC from Database A. Though we can not conclusively determine this was the case, one possible explanation for the increase in performance could be the multithreaded result processing and optimization techniques used automatically in JDV when working with source databases.

Use Case 2 - Virtual database analytical testing

When performing queries to a federated virtual database with JDV, database administrators do not face the challenges of migrating data, user-defined functions, or stored procedures. Improper migration or damage from the migration can introduce significant errors. In addition, keeping data and user-defined functions in place can potentially save database administrators time and labor.

To demonstrate query execution when choosing to federate your data sources and query JDV directly, we configured JDV with a virtual database that joined together data from Databases A, B, C, and D databases in such a manner that would allow us to perform the same queries. Using JMeter, we executed the six queries of varying complexity, data sources, and result set sizes for one user on a single client connection. Query B used JDV data pushdown to Database A. Figure 7 shows (1) Our JDV pushdown to Database A response times for each query from the previous scenario, and (2) response times for each query to the federated data in the virtual database of JDV.

Query	Single-user response times (seconds)	
	JDV with federated virtual database from four different RDBMS	JDV pushdown to one database (Database A)
A	0.081	0.016
B	12.708	56.012
C	0.398	0.165
D	0.208	0.074
E	32.359	4.803
F	8.923	78.938
Total	**54.677**	**140.008**

Figure 7: Response times for directly querying the federated virtual database of JDV with data from up to four different RDBS and JDV pushdown to one database, Database A.

Comments on use case 2

The goal of this test was to demonstrate how JDV could handle queries of varying complexity (for example, query results with many rows or many columns).The test offered the chance to check correctness of data, to measure latency introduced by JDV performing joins and aggregations, and to measure query response times for a single user.

Compared to the baseline response times, some queries had shorter response times with JDV in the architecture and others had longer response times. The total query response time was 61.7 percent better than the baseline total response time.

For queries B and F, response times were 78.4 and 88.8 percent better respectively when using JDV to query across multiple data sources simultaneously compared to baseline results.

One possible explanation for the reduction in response time in queries B and F could be that JDV pulled only relevant data into the JDV engine in a multithreaded fashion. The combination of concurrently streamed data and efficient join, sort, and aggregate algorithms in JDV could have allowed for faster response time. In addition, the JDV optimizer could have pushed down relevant data to the sources, which would mean that more processing of the query occurred at the source than at the JDV engine. Note that we can not state conclusively that this was the reason.

With a relatively small number of rows returned, introducing JDV into the solution added latency and increased response time for query A. As the number of rows returned increased, the difference in time greatly diminished or improved when JDV was introduced in the architecture. For queries C, D, and E, which had more returned rows than query A, combining the data sources with JDV also increased response times. This increase for some queries was due to the amount of query complexity and to the amount of data that needed to be retrieved from the sources into the JDV engine in real time prior to doing further query processing, such as joining the data from different databases, sorting (ORDER BY), and aggregating (GROUP BY) results.

Use Case 3 - Scaling the queries to a virtual database analytical workload

Using JMeter, we executed query G across four databases (Databases A, B, C, and D) with a varying number of concurrent users. Query G returned a large amount of data (20,166,673 bytes in 92,520 rows) as expected in a typical analytic workload. For this type of analytical workload, we did not expect to have more than 20 analytic users querying JDV simultaneously, so we varied the concurrent clients from one to 20 simultaneous users to simulate small, medium, and large analytics teams. Figure 8 shows the total query times when running the test continuously for 20 minutes.

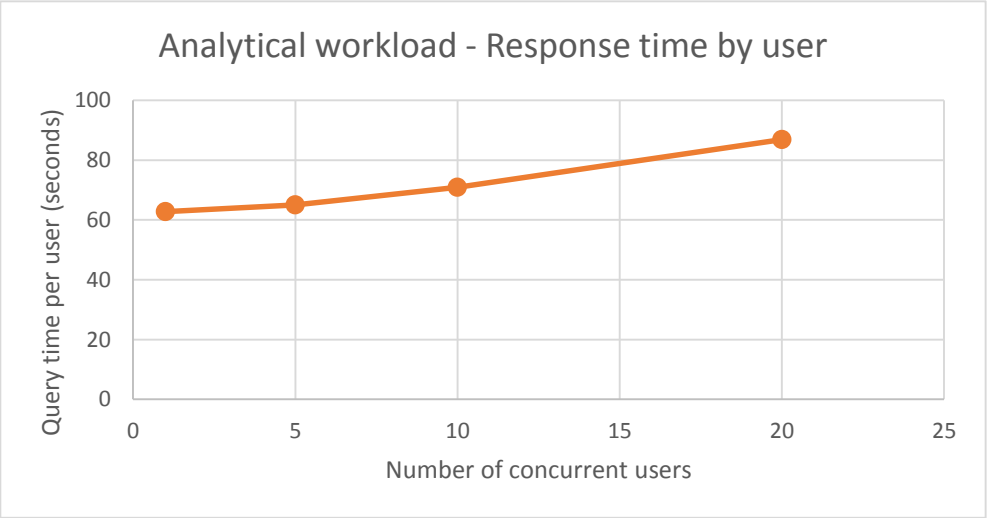


Figure 8: Analytical workload query times for an increasing number of concurrent users. Note: Line is used to guide the eye.

Comments on use case 3

As the number of users increased, latency increased slightly but the overall system resource consumption in terms of CPU load and memory consumption of the Java VM (JVM) stayed relatively proportional to the load. Increasing the processing batch size improved the processing of the query. With additional memory overhead, however, we did not observe any disk access by JDV that indicated the buffering of results to disk. JDV can effectively handle larger loads without severely degrading the system performance.

Use Case 4 - Scaling the queries to a virtual database OLTP workload

Using JMeter, we executed query H to the same federated virtual database in JDV using data from Databases A, B, C, and D. Query H performed a four-way join as did Query G above, but the result set size featured only a small number of rows. This small result set simulated transactional workloads with high volumes of concurrent users who would each execute small transactions, such as website usage. Figure 9 shows the number of concurrent users executing query H and corresponding throughput (queries per second). Note that each submitted query was unique, so result caching did not skew the results.

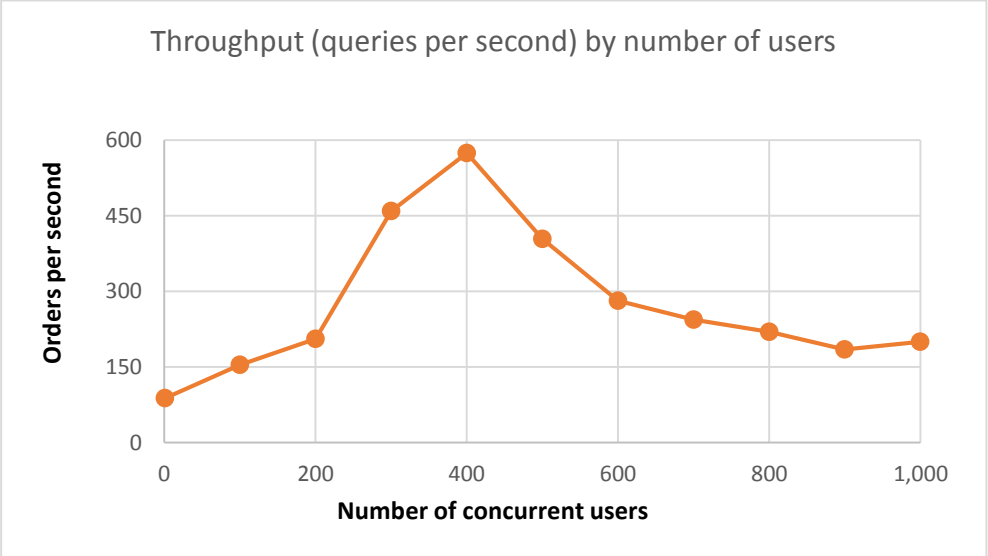


Figure 9: Throughput (queries per second) for an increasing number of concurrent users. Note: Line is used to guide the eye.

We performed this test with a new configuration of JDV that was appropriate for this OLTP query. Namely, we tested the OLTP performance four identical instances of JDV on the same server sharing the same amount of resources—each Java VM had 16 GB of RAM and 64 connections to each source database. We used the same set of OLTP queries for each JDV configuration.

Comments on use case 4

For tests with the four JDV instances, JVM configurations, such as large memory and thread management, helped create this improvement.[3]

Performance with four JDV instances peaked around 400 users, decreased as the load increased, and then stabilized to a constant rate. During this process, the CPU utilization was moderate across all machines (JDV and databases). I/O operations, JVM garbage collection, and source response time limited performance as CPU was not constrained with load.

To investigate further, we measured each source database's query times over the period of the test run and calculated the statistics of response times. We found that one source in particular was degrading in performance as load increased, and that limited overall throughput delivered through JDV.

The performance of JDV is therefore dependent upon the particulars of the integration scenario and the sources involved. If the sources are constrained, then adding JDV without additional caching techniques may not improve performance by itself. If you have a constrained source, then consider materialization and other caching strategies in JDV to improve system performance.[4]

[3] For more information, see Appendix C.

[4] To learn more about caching strategy, see JDV product documentation at https://access.redhat.com/documentation/en-US/Red_Hat_JBoss_Data_Virtualization/6.1/html/Development_Guide_Volume_5_Caching_Guide/index.html.

CONCLUSION

Without disrupting existing data and operations, your business can effectively use Red Hat JDV to query data in many environments, including analytic and OLTP workloads, both using multiple databases. Figure 10 summarizes our findings for the four use cases.

Use case	Description	Comments
Use case 1	Querying one Database A instance: Directly vs. pushdown using JDV	Queries using JDV produced response times with no measurable overhead compared to direct queries to the database without JDV.[5]
Use case 2	Querying a virtual database for an analytical workload	The queries to federated data from up to four data sources ran 61.7 percent faster than the baseline response times to a single data source. One possible explanation is that JDV acts as a logical data warehouse, accessing data from all sources in realtime, compared to the data residing in a single, physical data warehouse. We can not state conclusively that this was the reason.
Use case 3	Scaling the queries of an analytical workload	A 2X workload increase resulted in less than a 10 percent increase in response time.[6] As the number of users increased, latency increased slightly but the overall system resource consumption in terms of CPU load and memory consumption of the JVM stayed relatively proportional to the load. JDV effectively handled larger loads without severely degrading the system performance.
Use case 4	Querying a virtual database for an online transaction processing (OLTP) workload	There was a 272 percent increase in throughput when the number of concurrent users increased by 400 percent of the initial workload. Following this initial steady increase in performance as more concurrent users were added, the performance hit the constraints of the slowest data source. We then observed a steady 30 percent decline in throughput as we increased the number of concurrent users. Note that the JDV server did not show any sign of fatigue or stress.

Figure 10: Our findings for the four use cases.

The advantages for data virtualization abound for business agility by providing real time information across multiple, heterogeneous data sources without moving or copying any data, allowing businesses to respond quickly and accurately while reducing cost and data sprawl. Enterprises can use JDV without disrupting their current data and operations infrastructure to achieve their data integration and data abstraction goals. With an understanding of the use cases, a properly architected JDV server and set of virtual databases could meet performance expectations and in some cases, possibly improve performance.

[5] Actual results showed that queries through JDV performed 2 percent faster than queries directly to the data source. One possible explanation for the increase in performance could be the multithreaded result processing and optimization techniques used automatically in JDV when working with source databases. Note that we can not state conclusively that this was the reason.

[6] This number was calculated by looking at the increase from 5 to 10 concurrent users (2X) and the response increase of less than 10 percent. For each additional increment of users (10 to 15, and 15 to 20) the increase in response time remained around 10 percent.

APPENDIX A – THE SOFTWARE WE USED

Database software

- Database A, latest publicly available version: We can not publish the name of Database A due to EULA restrictions.
- Database B: PostgreSQL 9.2.10
- Database C: Microsoft SQL Server 2012
- Database D: MySQL Database 5.6.23

JBoss Data Virtualization

According to JBoss.org, "JBoss Data Virtualization is a data integration solution that sits in front of multiple data sources and allows them to be treated as a single source, delivering the right data, in the required form, at the right time to any application and/or user."[7] For more information, see www.redhat.com/en/technologies/jboss-middleware/data-virtualization and www.jboss.org/products/datavirt/overview/.

The benchmark testing tools: Apache JMeter and the Java OLTP query generators

JMeter is an open-source benchmark tool for testing web sites, web applications, and databases. When used to test databases, the benchmark queries the data source via a JDBC connector. For more information about JMeter, visit jmeter.apache.org.

To compile the JDBCClient application, we performed the following operation:

```
javac -classpath /root/jm/client/teiid-8.7.1.redhat-8-jdbc.jar JDBCClient.java
```

For this test, we ran this script (run.sh) as

```
sh run.sh 12 25 600000 # 12 instances of 25 threads each for 10 minutes
```

The file "run.sh":

```
JAVA_OPTS="-d64 -server -Xmx2G -Xmn1G"
JAVA_OPTS="$JAVA_OPTS -XX:+UseConcMarkSweepGC -XX:+UseParNewGC"


SQL="select s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address,
  s_phone, s_comment, c_name, o_orderdate, r_name
from
  /*+ makedep */ dbC.tpch.dbo.part, /*+ makedep */
  dbC.tpch.dbo.supplier, /*+ makedep */ dbC.tpch.dbo.partsupp,
  dbD.NATION, dbD.REGION, dbB.customer, /*+ makedep */
  dbA.orders, /*+ makedep */ dbA.lineitem
where
  (p_partkey = ps_partkey) and (s_suppkey = ps_suppkey) and (c_custkey =
  O_CUSTKEY) and (O_ORDERKEY = L_ORDERKEY) and (L_PARTKEY = p_partkey)
  and (L_SUPPKEY = s_suppkey) and (s_nationkey = N_NATIONKEY) and
  (N_REGIONKEY = R_REGIONKEY) and (c_custkey = ?)"

date
for i in $(seq 1 $1); do
   java $JAVA_OPTS -classpath /root/jm/client/teiid-8.7.1.redhat-8-jdbc.jar:. \
      -Dusername=OurUser -Dpassword=OurPassword JDBCClient "$2" "$3" "$SQL" &
done

wait
date
```

[7] Overview of Red Hat JBoss Data Virtualization www.jboss.org/products/datavirt/overview/

APPENDIX B – THE DATABASE QUERIES WE USED

Single-user DSS queries (A through F); multi-user analytic query G; multi-user OLTP query H

We generated the data using the TPC-H-like schema and its data-generation program. Red Hat Engineering adapted several of the TPC-H-like queries for these tests to demonstrate the capabilities of JDV. We considered eight total queries in three categories: six single-user DSS queries, one multi-user analytic query, and one multi-user OLTP query. Figures 11 through 14 list the queries and data sources used for each.

Query	SQL	Description	Data source

A	`SELECT *`
	`FROM customer`
	`WHERE c_custkey <= 100`
		Single data source	Database A

B	`SELECT s_acctbal, s_name, n_name, p_partkey, p_mfgr,`
	`  s_address, s_phone, s_comment, c_name, o_orderdate`
	`FROM part, supplier, partsupp, nation, region,`
	`  customer, orders, lineitem`
	`WHERE p_partkey = ps_partkey AND s_suppkey = ps_suppkey`
	`  AND c_custkey = o_custkey AND o_orderkey = l_orderkey`
	`  AND l_partkey = p_partkey AND l_suppkey = s_suppkey`
	`  AND p_size = 22 AND p_type LIKE '%TIN'`
	`  AND s_nationkey = n_nationkey`
	`  AND n_regionkey = r_regionkey`
	`  AND r_name = 'AFRICA' AND ps_supplycost < 20`
		Federated inner join; similar to TPC-H query Q2	Database A

C	`SELECT l_orderkey,`
	`  sum(l_extendedprice * (1 - l_discount)) AS revenue,`
	`  o_orderdate, o_shippriority`
	`FROM customer, orders, lineitem`
	`WHERE c_mktsegment = 'HOUSEHOLD' AND c_custkey =`
	`o_custkey`
	`  AND l_orderkey = o_orderkey`
	`  AND o_orderdate < '1995-03-01'`
	`  AND l_shipdate > '1995-03-01' AND c_custkey < 10000`
	`GROUP BY l_orderkey, o_orderdate, o_shippriority`
	`ORDER BY revenue DESC, o_orderdate`
		Federated inner join; similar to TPC-H query Q2	Database A

D	`SELECT c_custkey, count(o_orderkey) AS c_count`
	`FROM customer`
	`LEFT OUTER JOIN orders ON c_custkey = o_custkey`
	`WHERE o_comment NOT LIKE '%special%accounts%'`
	`  AND c_custkey < 10000`
	`GROUP BY c_custkey`
		Federated left outer join; similar to the subquery in TPC-H query Q13	Database A

E	`SELECT o_orderkey, o_orderdate, o_clerk,`
	`  c_custkey, c_name`
	`FROM orders`
	`JOIN customer ON c_custkey = o_custkey`
	`WHERE c_nationkey = 1 AND c_acctbal > 5`
	`  AND c_acctbal < 200`
	`  AND o_orderdate < {ts '1998-01-01 00:00:00' }`
		Federated dependent join	Database A

F	`SELECT c_custkey, c_name, c_phone`
	`FROM customer`
	`WHERE c_nationkey = 1 AND c_acctbal > 5`
	`  AND c_acctbal < 200`
	`UNION`
	`SELECT c_custkey, c_name, c_phone`
	`FROM CUSTOMER`
	`WHERE c_nationkey = 2 AND c_acctbal > 5`
	`  AND c_acctbal < 200`
		Federated union	Database A

Figure 11: Six single-user DSS queries of varying complexity , using only the Database A for the data source

Query	SQL	Description	Data sources
A	SELECT * FROM dbB.customer WHERE c_custkey <= 100	Single data source	Database B
B	SELECT s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_comment, c_name, o_orderdate FROM dbC.tpch.dbo.part, dbC.tpch.dbo.supplier, dbC.tpch.dbo.partsupp, dbD.nation, dbD.region, dbB.customer, dbA.orders, dbA.lineitem WHERE p_partkey = ps_partkey AND s_suppkey = ps_suppkey AND c_custkey = o_custkey AND o_orderkey = l_orderkey AND l_partkey = p_partkey AND l_suppkey = s_suppkey AND p_size = 22 AND p_type LIKE '%TIN' AND s_nationkey = n_nationkey AND n_regionkey = r_regionkey AND r_name = 'AFRICA' AND ps_supplycost < 20	Federated inner join; similar to TPC-H query Q2	Databases A, B, C, and D
C	SELECT l_orderkey, sum(l_extendedprice * (1 - l_discount)) AS revenue, o_orderdate, o_shippriority FROM dbB.customer, dbA.orders, dbA.lineitem WHERE c_mktsegment = 'HOUSEHOLD' AND c_custkey = o_custkey AND l_orderkey = o_orderkey AND o_orderdate < '1995-03-01' AND l_shipdate > '1995-03-01' AND c_custkey < 10000 GROUP BY l_orderkey, o_orderdate, o_shippriority ORDER BY revenue DESC, o_orderdate	Federated inner join; similar to TPC-H query Q2	Databases A and B
D	SELECT c_custkey, count(o_orderkey) AS c_count FROM dbB.customer LEFT OUTER JOIN dbA.orders ON c_custkey = o_custkey WHERE o_comment NOT LIKE '%special%accounts%' AND c_custkey < 10000 GROUP BY c_custkey	Federated left outer join; similar to the subquery in TPC-H query Q13	Databases A and B
E	SELECT o_orderkey, o_orderdate, o_clerk, c_custkey, c_name FROM dbA.orders JOIN dbB.customer ON c_custkey = o_custkey WHERE c_nationkey = 1 AND c_acctbal > 5 AND c_acctbal < 200 AND o_orderdate < {ts '1998-01-01 00:00:00' }	Federated dependent join	Databases A and B
F	SELECT c_custkey, c_name, c_phone FROM dbB.customer WHERE c_nationkey = 1 AND c_acctbal > 5 AND c_acctbal < 200 UNION SELECT c_custkey, c_name, c_phone FROM dbD.CUSTOMER WHERE c_nationkey = 2 AND c_acctbal > 5 AND c_acctbal < 200	Federated union	Databases B and D

Figure 12: The same six single-user DSS queries as in Figure 12, but updated to use a fully federated virtualize JDV database (four data sources).

Query	SQL	Description	Data sources

G	SELECT s_acctbal, s_name, n_name, p_partkey, p_mfgr,
	  s_address, s_phone, s_comment, c_name, o_orderdate
	FROM dbC.tpch.dbo.part, dbC.tpch.dbo.supplier,
	  dbC.tpch.dbo.partsupp, dbD.nation, dbD.region,
	  dbB.customer, dbA.orders, dbA.lineitem
	WHERE p_partkey = ps_partkey AND s_suppkey = ps_suppkey
	  AND c_custkey = o_custkey AND o_orderkey = l_orderkey
	  AND l_partkey = p_partkey AND l_suppkey = s_suppkey
	  AND p_size = 22 AND p_type LIKE '%TIN'
	  AND s_nationkey = n_nationkey
	  AND n_regionkey = r_regionkey
	  AND r_name = 'AFRICA' AND ps_supplycost < 20
		Federated inner join; identical to Query B	Databases A, B, C, and D

Figure 13: One multi-user analytic query to a fully federated virtualize JDV database (four data sources)..

Query	SQL	Description	Data sources

H	SELECT s_acctbal, s_name, n_name, p_partkey, p_mfgr,
	  s_address, s_phone, s_comment, c_name, o_orderdate,
	  r_name
	FROM
	  dbC.tpch.dbo.part, dbC.tpch.dbo.supplier,
	  dbC.tpch.dbo.partsupp, dbD.nation, dbD.region,
	  dbB.customer, dbA.orders, dbA.lineitem
	WHERE p_partkey = ps_partkey AND s_suppkey = ps_suppkey
	  AND c_custkey = o_custey AND o_orderkey = l_orderkey
	  AND l_partke = p_partkey AND l_suppkey = s_suppkey
	  AND s_nationkey = n_nation
	  AND n_regionkey = r_regionkey
	  AND (c_custkey = ${__Random(1, 150000000) })
		Federated inner join; adapted from Query B; note that the query driver generates a random customer identifier (to match c_custkey) for each query.	Databases A, B, C, and D

Figure 14: One multi-user OLTP query to a fully federated virtualize JDV database (four data sources).

Figure 15 shows the number of rows returned for each of the queries in Use case 1 and 2.

Query	Rows returned (Direct and JDV pushdown)
A	100
B	92,520
C	754
D	6,666
E	970,839
F	213,211
Total	**1,284,097**

Figure 15: Row counts returned for each of the queries in Use case 1 and 2.

APPENDIX C – SYSTEM CONFIGURATION INFORMATION

Figures 16 through 18 provide detailed configuration information for the test systems.

System	Database A server	Database B server
Power supplies		
Total number	2	2
Vendor and model number	DELTA DPS-750AB-21 A	DELTA DPS-750AB-21 A
Wattage of each (W)	750	750
Cooling fans		
Total number	8	8
Vendor and model number	San Ace 80 9G0812P1G09	San Ace 80 9G0812P1G09
Dimensions (h x w) of each	3-1/4" x 1-1/2"	3-1/4" x 1-1/2"
Volts	12 v	12 v
Amps	1.4 amps	1.4 amps
General		
Number of processor packages	2	2
Number of cores per processor	18	16
Number of hardware threads per core	2	2
System power management policy	Balanced	Balanced
CPU		
Vendor	Intel	Intel
Name	Xeon	Xeon
Model number	E5-2699 v3	E5-2698 v3
Stepping	C1	C1
Socket type	FCLGA2011-3	FCLGA2011-3
Core frequency (GHz)	2.30	2.30
Bus frequency	9.6 GT/s	9.6 GT/s
L1 cache	32 KB + 32 KB (per core)	32 KB + 32 KB (per core)
L2 cache	256 KB (per core)	256 KB (per core)
L3 cache	45 MB (shared)	40 MB (shared)
Platform		
Vendor and model number	Lenovo ThinkServer RD550	Lenovo ThinkServer RD550
Motherboard model number	70CX001YUX	70CW0003UX
BIOS name and version	PB1TS110(V1.10.0)	PB1TS110(V1.10.0)
BIOS settings	Default	Default
Memory module(s)		
Total RAM in system (GB)	128	128
Vendor and model number	SK Hynix HMA42GR7MFR4N-TF	SK Hynix HMA42GR7MFR4N-TF
Type	PC4-2133	PC4-2133
Speed (MHz)	2,133	2,133
Speed running in the system (MHz)	2,133	2,133
Timing/Latency (tCL-tRCD-tRP-tRASmin)	15-15-15-33	15-15-15-33
Size (GB)	16	16
Number of RAM module(s)	8	8

System	Database A server	Database B server
Chip organization	Double-sided	Double-sided
Rank	Dual	Dual
Hard disk		
Vendor and model number	Seagate ST600MM0006	Seagate ST600MM0006
Number of disks in system	6	6
Size (GB)	600	600
Buffer size (MB)	64	64
RPM	10K	10K
Type	SAS	SAS
Hard disk		
Vendor and model number	Seagate ST400FM0053	Seagate ST400FM0053
Number of disks in system	2	2
Size (GB)	400	400
Buffer size (MB)	N/A	N/A
RPM	SSD	SSD
Type	SAS	SAS
Disk controller		
Vendor and model	Lenovo ThinkServer RAID 720i	Lenovo ThinkServer RAID 720i
Controller cache	1 GB	1 GB
Controller driver	LSI MegaRAID SAS Driver 06.805.06.01-rc1	LSI MegaRAID SAS Driver 06.805.06.01-rc1
Operating system		
Name	Red Hat Enterprise Linux Server 7.1	Red Hat Enterprise Linux Server 7.1
Kernel	3.10.0-229.el7.x86_64	3.10.0-229.el7.x86_64
File system	xfs	xfs
Language	English	English
Ethernet		
Vendor and model number	Intel 82599ES 10-Gigabit Ethernet	Intel 82599ES 10-Gigabit Ethernet
Type	PCIe	PCIe
Driver	Intel 10 Gigabit PCI Express Network Driver, 4.0.1-k-rh7.1	Intel 10 Gigabit PCI Express Network Driver, 4.0.1-k-rh7.1
USB ports		
Number	6	6
Type	2.0	2.0

Figure 16: Configuration information for Database A & B servers.

System	Database C server	Database D server
Power supplies		
Total number	2	2
Vendor and model number	DELTA DPS-750AB-21 A	DELTA DPS-750AB-21 A
Wattage of each (W)	800	800
Cooling fans		
Total number	8	8
Vendor and model number	San Ace 80 9G0812P1G09	San Ace 80 9G0812P1G09
Dimensions (h x w) of each	3-1/4” x 1-1/2”	3-1/4” x 1-1/2”
Volts	12 v	12 v
Amps	1.4 amps	1.4 amps
General		
Number of processor packages	2	2
Number of cores per processor	10	10
Number of hardware threads per core	2	2
System power management policy	Balanced	Balanced
CPU		
Vendor	Intel	Intel
Name	Xeon	Xeon
Model number	E5-2690 v2	E5-2690 v2
Stepping	M1	M1
Socket type	FCLGA2011	FCLGA2011
Core frequency (GHz)	3.00	3.00
Bus frequency	8 GT/s	8 GT/s
L1 cache	32 KB + 32 KB (per core)	32 KB + 32 KB (per core)
L2 cache	256 KB (per core)	256 KB (per core)
L3 cache	25 MB (shared)	25 MB (shared)
Platform		
Vendor and model number	Lenovo ThinkServer RD540	Lenovo ThinkServer RD540
BIOS name and version	A1TS80A	A1TS80A
BIOS settings	Default	Default
Memory module(s)		
Total RAM in system (GB)	128	128
Vendor and model number	SK Hynix HMT42GR7AFR4C-RD	SK Hynix HMT42GR7AFR4C-RD
Type	PC3-14900	PC3-14900
Speed (MHz)	1,866	1,866
Speed running in the system (MHz)	1,866	1,866
Timing/Latency (tCL-tRCD-tRP-tRASmin)	13-13-13-34	13-13-13-34
Size (GB)	16	16
Number of RAM module(s)	8	8
Chip organization	Double-sided	Double-sided
Rank	Dual	Dual

System	Database C server	Database D server
Hard disk		
Vendor and model number	Seagate ST3450857SS	Seagate ST300MM0006
Number of disks in system	4	2
Size (GB)	450	300
Buffer size (MB)	16	64
RPM	15K	10K
Type	SAS	SAS
Hard disk		
Vendor and model number	n/a	Seagate ST900MM0006
Number of disks in system	n/a	4
Size (GB)	n/a	900
Buffer size (MB)	n/a	64
RPM	n/a	10K
Type	n/a	SAS
Hard disk		
Vendor and model number	n/a	Intel SSD DC S3700
Number of disks in system	n/a	2
Size (GB)	n/a	400
Buffer size (MB)	n/a	n/a
RPM	n/a	SSD
Type	n/a	SATA
Disk controller		
Vendor and model	LSI MegaRAID SAS 9270-8i	LSI MegaRAID SAS 9270-8i
Controller cache	1 GB	1 GB
Controller driver	Microsoft 6.3.9600.16384	LSI MegaRAID SAS Driver 06.805.06.01-rc1
Operating system		
Name	Microsoft Windows 2012 R2	Red Hat Enterprise Linux Server 7.1
Kernel/Build	9600	3.10.0-229.el7.x86_64
File system	ntfs	xfs
Language	English	English
Ethernet		
Vendor and model number	Intel 82599ES 10-Gigabit Ethernet	Intel 82599ES 10-Gigabit Ethernet
Type	PCIe	PCIe
Driver	Intel 3.9.58.9101	Intel 10 Gigabit PCI Express Network Driver, 4.0.1-k-rh7.1
USB ports		
Number	6	6
Type	2.0	2.0

Figure 17: Configuration information for Database C & D servers.

System	JBoss Data Virtualization server
Power supplies	
Total number	2
Vendor and model number	DELTA DPS-750AB-21 A
Wattage of each (W)	750
Cooling fans	
Total number	8
Vendor and model number	San Ace 80 9G0812P1G09
Dimensions (h x w) of each	3-1/4” x 1-1/2”
Volts	12 v
Amps	1.4 amps
General	
Number of processor packages	2
Number of cores per processor	18
Number of hardware threads per core	2
System power management policy	Balanced
CPU	
Vendor	Intel
Name	Xeon
Model number	E5-2699 v3
Stepping	C1
Socket type	FCLGA2011-3
Core frequency (GHz)	2.30
Bus frequency	9.6 GT/s
L1 cache	32 KB + 32 KB (per core)
L2 cache	256 KB (per core)
L3 cache	45 MB (shared)
Platform	
Vendor and model number	Lenovo ThinkServer RD550
Motherboard model number	70CX001YUX
BIOS name and version	PB1TS110(V1.10.0)
BIOS settings	Default
Memory module(s)	
Total RAM in system (GB)	128
Vendor and model number	SK Hynix HMA42GR7MFR4N-TF
Type	PC4-2133
Speed (MHz)	2,133
Speed running in the system (MHz)	2,133
Timing/Latency (tCL-tRCD-tRP-tRASmin)	15-15-15-33
Size (GB)	16
Number of RAM module(s)	8
Chip organization	Double-sided
Rank	Dual

System	JBoss Data Virtualization server
Hard disk	
Vendor and model number	Seagate ST3300657SS
Number of disks in system	4
Size (GB)	300
Buffer size (MB)	16
RPM	15K
Type	SAS
Disk Controller	
Vendor and model	LSI MegaRAID SAS
Controller cache	1 GB
Controller driver	LSI MegaRAID SAS Driver 06.805.06.01-rc1
Operating system	
Name	Red Hat Enterprise Linux Server 7.1
Kernel	3.10.0-229.el7.x86_64
File system	xfs
Language	English
Ethernet	
Vendor and model number	Intel 82599ES 10-Gigabit Ethernet
Type	PCIe
Driver	Intel 10 Gigabit PCI Express Network Driver, 4.0.1-k-rh7.1
USB ports	
Number	6
Type	2.0

Figure 18: Configuration information for the JBoss data virtualization server.

APPENDIX D – SOFTWARE CONFIGURATION INFORMATION

Configuring software by servers

Configuring all servers

We installed Red Hat Enterprise Linux 7 and its core packages on three database, the JDV, and the test-harness servers. We installed Microsoft Windows Server 2012 R2 on the fourth database server. After installation, we connected each to the 10GbE network, enabled NTP time synchronization, and disabled the firewall. We disabled SELinux and NetworkManager on the Linux servers.

Configuring the Red Hat JBoss Data Virtualization 6.1 server

We installed and configured JDV 6.1 as follows:

1. `yum install java-1.7.0-openjdk java-1.7.0-openjdk-devel`
2. Executed the JDV installer.

Configuring the Mysql database server

We installed and configured MySQL 5.6.23-3 as follows.

1. `wget` repo.mysql.com/mysql-community-release-el7-5.noarch.rpm
2. `yum localinstall mysql-community-release-el7-5.noarch.rpm`
3. `yum install mysql-server mysql-community-libs`
4. We modified the default configuration file /etc/my.cnf by adding the following settings to the mysqld stanza:

```
innodb_buffer_pool_size = 108G
innodb_log_file_size=2G
innodb_flush_method=O_DIRECT
innodb_io_capacity=2000
innodb_io_capacity_max=6000
innodb_lru_scan_depth=2000
#
skip-name-resolve
max_connect_errors = 100000
max_connections=256
query_cache_type=1
query_cache_size = 20M
query_cache_limit = 10G
key_buffer_size = 4G
```

5. We modified the system resource limits for the mysql user by adding the following to /etc/security/limits.conf:

```
mysql - nofile 8192
mysql - nproc  4096
```

6. `systemctl start mysqld`
7. `mysql_secure_installation`
8. `mysql -u root`

```
create user 'tpch'@'localhost" identified by 'tpch';
grant all privileges on *.* to  'tpch'@'%' identified by 'tpch' with grant
   option;
flush privileges;
```

Configuring the PostgreSQL database server

We installed PostgrSQL 9.2.10 as follows:

1. `yum install postgresql postgresql-libs postgresql-server`
2. `mkdir -p /u01/data`
3. `chown postgres:postgres /u01/data`
4. `su - postgres`
5. `initdb -D /u01/data`
6. `postgres -D /u01/data &`
7. `createddb tpch`
8. `createuser -U postgres -d -e -E -l -P -r -s  tpch`
9. We updated the following line to the database connection configuration file, /u01/data/gp_hba/conf:
   ```
   host all all 10.41.5.0/24 trust
   ```
10. We modified the database configuration by adding or updating the following to /u01/data/postgresql.conf:
    ```
    max_connections = 260
    shared_buffers = 32GB
    work_mem = 832MB
    maintenance_work_mem = 1GB
    wal_buffers = 32MB
    checkpoint_segments = 64
    checkpoint_completion_target = 0.9
    effective_cache_size = 96GB
    default_statistics_target = 1000
    constraint_exclusion = on
    log_checkpoints = on
    log_connections = on
    ```
11. We restarted the database with this new configuration:
    ```
    pg_ctl stop -D /u01/data; pg_ctl start -D /u01/data
    ```

Configuring the Database A

We installed and configured Database A following this methodology:

1. We installed addition RPMs from the standard Red Hat Enterprise Linux repositories.
2. We created a non-root laccount for the database administrator.
3. We modified system resources for the database administrator, similar to step 5 in the cionfiguring MySQL database section.
4. We modified the system resources in /etc/sysctl.conf to enable the use of huge pages and to increase the resourcdes available to the System V IPC subsystem.
5. We installed the database software per the vendor's documentation.

Configuring the test-harness database server

We installed Apache JMeter 2.13 and configured it to use the JDBC drivers for JDV (`teiid-8.7.1.redhat-8-jdbc.jar`) and Database A. We installed OpenJDK 1.7.0 for the Java VM environment.

Configuring the Microsoft SQL Server server

We installed Microsoft SQL Server 12 (64 bit) with the default parameters, and created a database instance named "tpch". We configured SQL Server to use a maximum degree of parallelism to 16, and Windows and SQL authentication. We created a database user "tpch" and granted it access to the database instance "tpch".

Creating the database schema and generating data

We used the TPC-H-like data generation program available from tpch.org. We generated one dataset with a scale factor of 1,000 (approximately 1 TB) to be loaded into each database. We exported the directory containing the data tables via both NSF and CIFS.

After creating the tables on each database, we loaded the data into each database as follows:

1. Database A:
   We ran one SQL script to populate the tables and create indices.
2. Database B: PosrgreSQL
   a. We mounted the shared directory at /mnt
   b. We made a named pip file: `mkfifo --mode=0666 /tmp/fifo`
   c. For each table, we stripped the trailing delimiter from its data file, sent it to the named pipe, and populated the database table from the named pipe.

```
for table in partsupp part supplier region nation orders customer lineitem; do
  echo Working of table $table
  sed -i 's/|$//' /mnt/customer.tbl > /tmp/fifo &
  psql -U tpch -d tpch -c 'copy '$table' from '\''/tmp/fif\'' with delimiter as
   '\''|\''
done
```

3. Database C: Microsoft SQL Server
   a. We mounted the shared data director at Z:
   b. We created and ran a t-sql script to load the data

```
BULK INSERT partsupp FROM 'Z:\tpch_2_17_0\partsupp.tbl' WITH (TABLOCK,
   DATAFILETYPE='char', CODEPAGE='raw', FIELDTERMINATOR = '|', rowterminator =
   '0x0a');
BULK INSERT part FROM 'Z:\tpch_2_17_0\part.tbl' WITH (TABLOCK,
   DATAFILETYPE='char', CODEPAGE='raw', FIELDTERMINATOR = '|', rowterminator =
   '0x0a');
BULK INSERT supplier FROM 'Z:\tpch_2_17_0\supplier.tbl' WITH (TABLOCK,
   DATAFILETYPE='char', CODEPAGE='raw', FIELDTERMINATOR = '|', rowterminator =
   '0x0a');
BULK INSERT region FROM 'Z:\tpch_2_17_0\region.tbl' WITH (TABLOCK,
   DATAFILETYPE='char', CODEPAGE='raw', FIELDTERMINATOR = '|', rowterminator =
   '0x0a');
BULK INSERT nation FROM 'Z:\tpch_2_17_0\nation.tbl' WITH (TABLOCK,
   DATAFILETYPE='char', CODEPAGE='raw', FIELDTERMINATOR = '|', rowterminator =
   '0x0a');
BULK INSERT orders FROM 'Z:\tpch_2_17_0\orders.tbl' WITH (TABLOCK,
   DATAFILETYPE='char', CODEPAGE='raw', FIELDTERMINATOR = '|', rowterminator =
   '0x0a');
BULK INSERT customer FROM 'Z:\tpch_2_17_0\customer.tbl' WITH (TABLOCK,
   DATAFILETYPE='char', CODEPAGE='raw', FIELDTERMINATOR = '|', rowterminator =
   '0x0a');
BULK INSERT lineitem FROM 'Z:\tpch_2_17_0\lineitem.tbl' WITH (TABLOCK,
   DATAFILETYPE='char', CODEPAGE='raw', FIELDTERMINATOR = '|', rowterminator =
   '0x0a');
```

4. Database D: MySQL
   a. We mounted the shared directory at /mnt
   b. We made a named pip file: `mkfifo --mode=0666 /tmp/fifo`
   c. For each table, we sent it to the named pipe, and populated the database table from the named pipe. For example,

```
cat mnt/customer.tbl > /tmp/fifo &
# in the mysql shell
mysql> LOAD DATA INFILE '/tmp/fifo' INTO TABLE CUSTOMER fields terminated by '|'
   lines terminated by '\n';
```

APPENDIX E – ADDITIONAL INFORMATION FOR TRANSACTIONAL WORKLOAD TESTING

For large page support in JVM, please visit the following pages:

- https://access.redhat.com/documentation/en-US/JBoss_Enterprise_Application_Platform/5/html/Performance_Tuning_Guide/sect-Performance_Tuning_Guide-Java_Virtual_Machine_Tuning-Large_Page_Memory.html
- https://access.redhat.com/documentation/en-US/JBoss_Enterprise_Application_Platform/5/html/Performance_Tuning_Guide/sect-Performance_Tuning_Guide-Java_Virtual_Machine_Tuning-Large_Page_Memory.html

Configuring the Java VM for Red HAT JDV

We used the following Java VM parameters for each instance of JDV for the OLTP tests. In particular, we switched to G1 garbage collector.

```
-server
-Xmx16G
-Xms16G
-XX:+UseLargePages
-XX:MaxPermSize=512m
-XX:+UseG1GC
-XX:InitiatingHeapOccupancyPercent=0
-XX:+DisableExplicitGC
-XX:MaxGCPauseMillis=200
-XX:GCPauseIntervalMillis=5000
```

ABOUT PRINCIPLED TECHNOLOGIES

Principled Technologies, Inc.
1007 Slater Road, Suite 300
Durham, NC, 27703
www.principledtechnologies.com

We provide industry-leading technology assessment and fact-based marketing services. We bring to every assignment extensive experience with and expertise in all aspects of technology testing and analysis, from researching new technologies, to developing new methodologies, to testing with existing and new tools.

When the assessment is complete, we know how to present the results to a broad range of target audiences. We provide our clients with the materials they need, from market-focused data to use in their own collateral to custom sales aids, such as test reports, performance assessments, and white papers. Every document reflects the results of our trusted independent analysis.

We provide customized services that focus on our clients' individual requirements. Whether the technology involves hardware, software, Web sites, or services, we offer the experience, expertise, and tools to help our clients assess how it will fare against its competition, its performance, its market readiness, and its quality and reliability.

Our founders, Mark L. Van Name and Bill Catchings, have worked together in technology assessment for over 20 years. As journalists, they published over a thousand articles on a wide array of technology subjects. They created and led the Ziff-Davis Benchmark Operation, which developed such industry-standard benchmarks as Ziff Davis Media's Winstone and WebBench. They founded and led eTesting Labs, and after the acquisition of that company by Lionbridge Technologies were the head and CTO of VeriTest.

Principled Technologies is a registered trademark of Principled Technologies, Inc.
All other product names are the trademarks of their respective owners.

Disclaimer of Warranties; Limitation of Liability:
PRINCIPLED TECHNOLOGIES, INC. HAS MADE REASONABLE EFFORTS TO ENSURE THE ACCURACY AND VALIDITY OF ITS TESTING, HOWEVER, PRINCIPLED TECHNOLOGIES, INC. SPECIFICALLY DISCLAIMS ANY WARRANTY, EXPRESSED OR IMPLIED, RELATING TO THE TEST RESULTS AND ANALYSIS, THEIR ACCURACY, COMPLETENESS OR QUALITY, INCLUDING ANY IMPLIED WARRANTY OF FITNESS FOR ANY PARTICULAR PURPOSE. ALL PERSONS OR ENTITIES RELYING ON THE RESULTS OF ANY TESTING DO SO AT THEIR OWN RISK, AND AGREE THAT PRINCIPLED TECHNOLOGIES, INC., ITS EMPLOYEES AND ITS SUBCONTRACTORS SHALL HAVE NO LIABILITY WHATSOEVER FROM ANY CLAIM OF LOSS OR DAMAGE ON ACCOUNT OF ANY ALLEGED ERROR OR DEFECT IN ANY TESTING PROCEDURE OR RESULT.

IN NO EVENT SHALL PRINCIPLED TECHNOLOGIES, INC. BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH ITS TESTING, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL PRINCIPLED TECHNOLOGIES, INC.'S LIABILITY, INCLUDING FOR DIRECT DAMAGES, EXCEED THE AMOUNTS PAID IN CONNECTION WITH PRINCIPLED TECHNOLOGIES, INC.'S TESTING. CUSTOMER'S SOLE AND EXCLUSIVE REMEDIES ARE AS SET FORTH HEREIN.