WHITE PAPER

GridGain

Adding Speed and Horizontal Scale to PostgreSQL

In-Memory Computing Options for PostgreSQL Server Deployments

PostgreSQL is one of the most popular open source databases, and one of the most fragmented. There are over thirty different distributions and products built on PostgreSQL. These products give companies many options: almost too many to choose from when growing their PostgreSQL deployments. These options also have their limitations. New business needs and performance demands have pushed many applications and their underlying RDBMSs beyond their architectural limits. In many cases the issues cannot be remedied by just fixing the database layer.

Part of the challenge is scalability. Over the last decade, the adoption of digital business, IoT and other new technologies has increased query and transaction loads 10-1000x, and the amount of data collected by 50x.

Another major challenge is speed. Customer-facing web and mobile apps, and their underlying APIs, all require sub-second roundtrip latencies. This low latency requirement is impossible to support given all the network hops, software layers, data queries and joins that occur between the new customer apps and underlying applications and databases. In addition, the amount of data needed for different analytics and other computing types--including machine and deep learning--has become too big to move quickly enough over the network.

Another challenge lies in the development of the applications themselves. New business initiatives (such as changes to improve the end-to-end customer experience) require the delivery of new capabilities in days or weeks rather than months or years. But most existing applications are inflexible to change. It takes months just to deliver minor changes. Most applications also do not support many newer technologies such as streaming analytics or machine and deep learning, which are required to help streamline, automate, and improve real-time processes.

This white paper explains what your options are for adding speed, scale, and agility to end-to-end IT infrastructure—both from PostgreSQL-centric vendors and from other open source and third-party products. It also explains how to evolve your architecture over time to both increase speed and scale and help create flexible IT infrastructure that supports digital business and other new technology initiatives.

THE CHALLENGE WITH SPEED, SCALE, AGILITY, AND AUTOMATION

The latest variations of PostgreSQL and their extensions cannot deliver the performance and scalability required by changes in the demands placed on deployments, including:

- The adoption of new web, mobile, and other self-service channels
- The addition of personalization and other automation
- The use of new types of data

These new demands require speed and scale not just at the database, but at other system layers as well.

Over the last decade, these innovations have led to query and transaction volumes growing 10x to 1000x. They have resulted in 50x more data about customers, products, and interactions. They have also shrunk customers expected end-to-end response times from days or hours to seconds or less. The required roundtrip latency from a mobile device calling an API that accesses applications over the network—which in turn access PostgreSQL variations over the network—must always be less than one second.

The existing architecture cannot deliver with existing applications and databases. For one, vertical scalability is not a long-term option: the 10-1000x growth rates that show no signs of slowing down are faster than Moore's Law. Even if at the outset more is spent on hardware, eventually no single hardware server will be able to keep up with the growth. Latency is also a major issue. The combination of multiple network hops and queries that involve large data sets makes achieving the required speed and scalability with the current architecture practically impossible. As one architect put it, "You can't violate the laws of physics." The system needs lower end-to-end network latency, not just database latency.

Even if databases alone could address the speed and scale challenges, the applications themselves introduce a third major challenge. Customers have come to expect rapid change. The new innovators that are disrupting just about every business model deliver new capabilities in weeks or days. Today's applications are not that flexible: it can take months or years to develop and deploy changes.

Then there is the final challenge: automation. Streamlining a process, creating a one-click shopping experience, or watching for issues and proactively fixing them all require real-time intelligence and automation. The architecture of existing applications and databases separates online transactions processing (OLTP) from online analytical processing (OLAP), data warehousing, and business intelligence. OLTP systems cannot take the additional load and were not designed for analytics. OLTP applications usually only solved one problem and only used a portion of the data about the business. To create a single view of the business for analytics, companies must Extract, Transform and Load (ETL) data into a warehouse, cleanse the data, enrich it, and make it consistent

across different types of data—all to make reporting and ad hoc analytics work. When these systems were created, neither the freshness of the data nor the speed of reporting was as important as the accuracy of the result.

Now the analytics and the decisions must happen in real-time (often during a transaction) to deliver a great sub-second experience. That is impossible to do with existing applications. The only way to run analytics that quickly is to perform them on the same data in the same place as the transactions. Most applications, however, do not include analytics or technologies such as machine or deep learning that help with decision automation.

THE EVOLUTION TOWARDS IN-MEMORY COMPUTING

Many innovators have been able to address all the speed, scale, agility and automation challenges with an architecture that Gartner coined Hybrid Transactional/Analytical Processing (HTAP). One of the most common foundations for HTAP is in-memory computing. One of the most common in-memory computing platforms is Apache Ignite, along with its commercial supported version, GridGain.

With Apache Ignite, these companies were able to adopt in-memory computing incrementally, one project at a time. Most companies started by implementing Ignite as an in-memory data grid (IMDG) in-between existing databases and applications rather than buying additional database licenses or adding new database hardware. This method offloaded reads from the databases, giving the databases substantially more room for growth on their existing hardware and lowering latency. Of of the most common such databases is PostgreSQL. According to the annual in-memory computing survey, nearly 40% of participants use or expect to use Apache Ignite or GridGain with PostgreSQL.

These IMDG deployments also gave them a way to evolve around existing applications. They could be more agile by unlocking the data inside their applications for use by another project. For each project, as more data was added into their common in-memory computing infrastructure, the combined data could be used to:

- Perform real-time analytics
- Add speed and scale for new APIs
- Process new types of data with streaming analytics
- Help with automation using machine and deep learning

These deployments also improved business agility, as new workloads were easy to add without impacting existing applications by adding more nodes as needed. Ignite is also cloud native, which makes it straightforward to use in a DevOps environment.

OPTIONS FOR ADDING SPEED AND SCALE TO POSTGRESQL

Adopting HTAP and in-memory computing is the right long-term approach for adding speed, scale, agility, and digitally transforming into a real-time business. But there are also other short- and long-term options for helping with speed and scale.

PostgreSQL has been around for over 30 years. It started in 1986 as the POSTGRES project. A SQL interpreter was added in 1994 after several iterations and uses, and it became a Berkeley open-source project in 1995. When it became clear that Postgres95 did not have staying power as a name, it became PostgreSQL 1.0 in 1996. The releases continued with 6.0 in 1997, 7.0 in 2000, 8.0 in 2005, 9.0 in 2015, 10 in 2017, and 11 the end of 2018.

While there are other open source databases, PostgreSQL is one of the most popular. When Oracle acquired Sun (which had acquired MySQL AB in 2008), a fork of MySQL was made and MariaDB was born. This fragmented the MySQL market. While some MySQL customers moved to MariaDB, other MySQL customers who preferred "pure" open source migrated to PostgreSQL.

Today, companies that rely on PostgreSQL have almost too many options from the PostgreSQL community. There are many options that address different challenges with scale; and not nearly enough that adequately address challenges with speed. There are also other open source and third-party options that can address these challenges outside of the database. This white paper provides a comparison of all the options for improving scale, as well as guidelines for how to move towards in-memory computing over time to improve both speed and scale together.

First, we will compare the main PostgreSQL forks available to add speed and scale, along with recommendations for when to choose each option. Options evaluated include:

- For speed: pgmemcache and the column store extension
- For OLTP scale: PostgreSQL replication, Citus Data, Postgres-XL, Second Quadrant, EnterpriseDB, and Amazon RDS for Postgres
- For OLAP/analytical scale: many of the traditional real-time data warehouse products including Greenplum (now open source), as well as cloud options such as Amazon Aurora.

GridGain

LOWERING LATENCY: IN-MEMORY COLUMN STORE AND PGMEMCACHE

There are only a few options for lowering the latency of a PostgreSQL deployment. The first is simply to use a RAM disk for storage and attach it. Another is to look at UNLOGGED tables, if the table is significantly smaller than available RAM. Then there are two open source options: the in-memory column store and pgmemcache.

The in-memory column store is a drop-in extension to PostgreSQL that stores data as columns in RAM in shared buffers for the best analytical performance. The downsides are slower transactions and, more importantly, instabilities from shared buffers that were not really designed for this type of usage. Data can be lost even with the disk-based extensions.

The second option is pgmemcache. Basically, it adds user-defined functions so that an application can use it in the context of PostgreSQL. You must change your applications so that pgmemcache can manage the cache. It is important to note that pgmemcache has many of the same limitations as Memcached. While pgmemcache offloads reads from PostgreSQL and lowers read latency, it does not easily scale horizontally. Second, pgmemcache still requires an application to fetch data from the cache over a network. This does not seem like a big issue, but most corporate networks can at most move one gigabyte per second across all servers. Companies are increasingly hitting that limit.

In short, there is no perfect option within PostgreSQL for lowering latency.

OPTIONS FOR SCALING CORE POSTGRESQL VERTICALLY AND HORIZONTALLY

To improve scale, scaling vertically might seem like a good option in the short term. Like most other databases, PostgreSQL does scale vertically. But it is not a cost effective, or (in several cases) a viable long-term option. Data is growing too fast for hardware and the capacity of a single machine or network. Hardware costs explode once they exceed the capacity of current commodity hardware.

Horizontal scalability is the only viable long-term option. Traditional horizontal scaling solutions such as replication can help with read scalability, but they do not help with scaling the size of the database or with write scalability. The best answer for both is to create a distributed database by horizontally partitioning the data, SQL, and other computing.

Core PostgreSQL provides support for sharding data manually by key ranges, and for aggregate push downs. This may be enough for some deployments. But it is still largely a manual job to manage the partitions and replication for adding read scalability and failover, and rebalancing is hard. Sharding may not work well for managing many tables together supporting different workloads. In general, adopting a commercial PostgreSQL option, adding horizontal scale at another layer, or making changes to the application are better longer-term choices.

There are several commercial PostgreSQL offerings. Amazon offers RDS for PostgreSQL, for example, which makes it easier to manage features like replication and failover. Azure and Google have similar offerings. Amazon Aurora is compatible with PostgreSQL and has horizontal scalability for online transactional processing (OLTP) applications. Amazon Redshift is based on PostgreSQL and has online analytical (OLAP) processing).

There are a host of other vendors with their own OLAP implementations of PostgreSQL, including:

- HP Vertica
- Teradata (Aster Data)
- IBM Netezza
- Pivotal Greenplum (which contributed their implementation to open source)

Most of these vendors optimized for real-time ETL- or stream-based analytics and use columnar rather than row-based storage for the data. This white paper does not offer a comparison of all these products. It focuses on the various options for adding speed and scale to OLTP and HTAP workloads. It is very important to note that OLAP and OLTP are very different with PostgreSQL and should not be compared. Do not try to use an OLTP-optimized version of PostgreSQL for OLAP, or vice versa.

The following is a comparison of the main open source and commercial options that you can add to PostgreSQL.

Horizontal Read Scalability: PostgreSQL Replication

Unlike several other databases, core PostgreSQL does provide hot standby replication using streaming from the write-ahead logs (WAL). The hot standby servers can handle reads as well. They are asynchronous, however, so under certain conditions writes might not immediately be visible. PostgreSQL also has a "warm standby option where a server is used only on failover.

While replication does work, there are some challenges with failover both on the client and server side. Core PostgreSQL does not come with built-in monitoring and failover. By

default, PostgreSQL needs to promote the secondary to be the primary and make sure clients write to exactly one primary. PostgreSQL clients also connect to a single endpoint. There is no automatic failover. As a result, when a primary fails the clients continue to retry and fail. In addition, whenever a secondary node must be recreated, unless using a third party vendor, you need to recreate the instance by replaying the entire history. You can either work out your own failover and recovery architecture or choose a vendor that simplifies it.

Horizontal Read Scalability: Pgpool-ii

Pgpool-ii is a middleware load balancer that is placed in between PostgreSQL and the applications to make a replicated, highly available cluster look like a single PostgreSQL server to the application. Pgpool-ii provides read scalability by transparently handling all the routing, load balancing, and failover for the applications. To the applications, Pgpool-ii looks like the PostgreSQL database. Pgpool-ii does not help with write scalability in any way. It is not recommended for use as the replication mechanism (as noted by EnterpriseDB, a company that provides commercial support for Pgpool-ii and PostgreSQL).

Horizontal Read and Write Scalability: EnterpriseDB

One such commercial option that does simplify replication and failover is EnterpriseDB. EnterpriseDB was founded in 2004 and employs several leading contributors to PostgreSQL. The EDB Postgres Platform includes EDB Postgres Advanced Server and a set of tools for business-critical deployments. One of the tools is EDB Postgres Replication Server (EPRS). It provides heterogeneous, multi-master bi-directional replication between EDB Postgres Advanced Server, PostgreSQL, Oracle and SQL Server.

While Replication Server does work well, there are several limitations. First, it is eventually consistent, relying on WAL-based asynchronous replication for PostgreSQL and EDB Postgres, and triggers that are outside of the transactional context for Oracle and SQL Server. This means that transactions are not always truly guaranteed, though it does help automate any compensating transactions for conflicting writes. Second, for Oracle and SQL Server the trigger-based replication adds a significant load compared to WAL-based replication. This extra load might require additional server hardware and Oracle or SQL Server licenses. Third, while you can do (basic) range partitioning for tables, it is not automatic or efficient by default. You must manage partitioning manually and make sure partitions work properly with aggregate pushdowns across partitioned tables. In addition, you also must rebalance manually. Replication becomes more complex as well.

EnterpriseDB also used to have Infinite Cache, which acted as an application-transparent horizontally scalable write-through cache that loaded and cached data as needed. This technology was deprecated as of Release 8.2.

Distributed Read Scalability: Postgres-BDR

2ndQuadrant, founded in 2001 by one of the leading contributors to PostgreSQL, actively contributes to PostgreSQL and provides a host of tools, support, and services for PostgreSQL deployments. One of their main open source contributions is Postgres-BDR, or bi-directional replication. Like EnterpriseDB, BDR provides bi-directional replication. Unlike EnterpriseDB, it does not provide bi-directional integration with Oracle or SQL Server. But it does provide a choice of immediate ("eager") or eventual ("fast") consistency, whereas EnterpriseDB only supports eventual consistency.

Postgres-BDR also supports both replication and sharding. Each table can be defined as replicated (where each node has a full copy of the data), or sharded/partitioned (where each node has its own partition of the data with optional copies across nodes for redundancy). Data placement of partitions can be based on hash tables or user-defined. Queries can also be run using aggregate pushdowns to help minimize network traffic.

Distributed PostgreSQL: Postgres-XL

2ndQuadrant also helped create Postgres-XL, a horizontal scalable version of PostgreSQL that is "very closely compatible with PostgreSQL" according to 2ndQuadrant. The first question one should ask is why does 2ndQuadrant have both Postgres-BDR and Postgres-XL (since both seem to offer sharding)? The short answer is that while Postgres-BDR has basic sharding, it is not a distributed database and would be very hard to manage as a distributed database. Postgres-XL includes several optimizations for multi-node distributed queries (for massively parallel processing or MPP) including:

- Replication to help scale reads
- Dynamic data redistribution and balancing
- Distributed transaction processing across partitions

For transactions, Postgres-XL acts as a global XA coordinator across the nodes.

That said, there are some limitations. First, Postgres-XL is a fork of PostgreSQL and does not claim to be 100% compatible. Second, while it can now have a warm standby per

node, if a partition is lost it takes time to recover. Third, even though it does have a global transaction coordinator that supports distributed transactions, it is designed for and only tested for asynchronous (optimistic) transactions, not synchronous (pessimistic) transactions. In addition, the global transaction coordinator adds overhead, and each query or transaction can only use one core per node—which results in less scalability.

Distributed PostgreSQL: Citus Data

The other leading choice for truly distributed PostgreSQL is Citus Data, which was acquired by Microsoft in January 2019. Citus is available as an open source or Enterprise version on-premise, or on Azure or AWS as a service. It is an extension to PostgreSQL that adds a cluster coordinator and worker nodes. The coordinator node is responsible for managing distributed SQL, which for queries involves transforming queries into subqueries, sending the queries across the nodes, and aggregating the results. Citus supports massively parallel processing (MPP) for SQL by collocating data from related tables using their foreign key. It also supports zero-downtime rebalancing of shards as nodes are added to the cluster. Citus has had strong support for transactions within a single shard. As of release 7.1 in 2017 it now also supports distributed synchronous transactions and handles several distributed lock conditions across shards.

While Citus has many of the benefits of a distributed database, and offers the best solution for distributed PostgreSQL, it also has the limits of a being just a good, distributed PostgreSQL database. First, the options for lowering latency are limited when compared to an in-memory database. While Citus can scale out horizontally and use the combined RAM across a cluster, each node is still a PostgreSQL disk-based database. Using shared buffers to cache working data in RAM is an option, but it is not a memory-first architecture.

Second, Citus is a PostgreSQL-only solution. It does not, like most in-memory data grids, allow for merging data across heterogeneous databases for new uses, such as for building new APIs or accessing application data during stream processing. Many of the newer projects require accessing existing multiple databases, and most companies have not standardized on PostgreSQL. There are a host of other databases as well.

Third, while Citus partitions data for distributed SQL and collocates SQL with data for MPP at scale, it does not partition the data and collocate Java, .NET, C++ or any other code. Collocation is a critical component of scaling out a distributed database for those use cases where the data is too big to move over the network. It not only requires code distribution as a compute grid. It requires partitioning the data the right way to minimize network traffic for the specific code. If a company has any such use cases they would need to move the data from Citus into another store to support their Java, .NET, C++ or other compute tasks.

ADDING SPEED AND SCALE WITH IN-MEMORY COMPUTING

The other main option for adding speed and scale is to use in-memory computing. The end goal of in-memory computing is to move data into memory for speed, and to use a combination of a shared nothing architecture and MPP for linear, horizontal scale for all data-intensive workloads. HTAP needs both existing data in relational databases and new data, such as streaming data from Web interactions or devices, or social data that helps it understand customer preferences and relationships.

The most common first step is the use of in-memory computing as an IMDG to existing applications, for two reasons. First, an IMDG adds in-memory speed and horizontal scalability that is more cost-effective in the longer term than scaling up with expensive hardware.

Do the math. Add up all the expected read and write scalability needs for the next 3-5 years. Then figure out your long term options. Most companies discover the following: they can either spend the money now on expensive hardware like Exadata, and then must implement an IMDG in the future or add the IMDG now and slowly grow it to the same size in the future assuming no other uses.

Second, an IMDG unlocks existing data for new uses, such as for real-time analytics, HTAP, streaming analytics or machine and deep learning. To support all these projects requires other capabilities, namely:

- IMDB support for storing and managing new types of data alongside existing data
- Streaming analytics support, including integration with other streaming technologies like Apache Kafka and Spark
- Machine and deep learning support

Again, do the math. Identify the projects that can be achieved with existing data accessible in memory and add up the ROI over those 3-5 years. That is money lost without an IMDG as part of a broader in-memory computing platform. The ROI on those additional projects should be added to help decide between different in-memory computing technologies and other options.

HOW AN IMDG ADDS SPEED AND SCALE, AND UNLOCKS DATA

An IMDG adds speed and scale by sitting in-between applications and databases, in the path of all reads and writes. It stores all data in-memory and keeps the data up to date by supporting a read-through/write-through cache pattern. It receives all writes, writes to memory, and then passes it on to the database as a transaction. If the database transaction succeeds, the IMDG commits to memory as well. Since this keeps all data in the IMDG in sync with the database, the IMDG can handle all reads directly. This lowers latency for reads because the data is accessed directly from RAM, not a disk-based database. An IMDG also adds scale by offloading all read workloads from the database. Most IMDGs can scale horizontally on commoditized hardware to handle increased read loads without putting additional loads on the database. This is much less expensive than buying specialized database hardware.

The easiest way to slide an IMDG in-between an application and PostgreSQL is for the IMDG to support PostgreSQL. If it does not support PostgreSQL, then write new code that replaces the SQL with a key-value API, and more code for the IMDG to access PostgreSQL.

Most IMDGs also provide some form of massively parallel processing (MPP) where they divide up data into smaller sets across nodes and collocate code with the data (like Hadoop). MPP allows horizontal scalability of both the data and computing, like the way MapReduce or Spark work. If the data is partitioned so that the computing has all the data it needs on each node, then the data does not need to be fetched over the network. This approach helps eliminate one of the most common performance bottlenecks in big data analytics and general big data computing, the network.

A PostgreSQL database does not support MPP. If moving data over the network is part of the performance issue, scaling PostgreSQL will not solve the problem. Also, part of the reason for adding an IMDG is to unlock data that is in PostgreSQL, to be able to use the data in new projects, including HTAP, without overloading PostgreSQL or requiring a hardware upgrade. PostgreSQL, and a database in general, does not support real-time analytics or high performance computing at scale. It doesn't support Spark or other streaming analytics technologies. It does not support general-purpose machine or deep learning. All of these rely on MPP.

APACHE IGNITE AND THE GRIDGAIN IN-MEMORY COMPUTING PLATFORM

GridGain is the leading in-memory computing platform for real-time business. It is the only enterprise-grade, commercially supported version of the Apache® Ignite™ (Ignite) open source project. GridGain includes enterprise-grade security, deployment, management, and monitoring capabilities which are not in Ignite, plus global support and services for business-critical systems. GridGain Systems contributed the code that became Ignite to the Apache Software Foundation and continues to be the project's lead contributor.

GridGain and Ignite are used by tens of thousands of companies worldwide to add in-memory speed and unlimited horizontal scalability to existing applications, and then add HTAP to support new initiatives to improve the customer experience and business outcomes. With GridGain, companies have:

- Improved speed and scalability by sliding GridGain in-between existing applications and databases as an IMDG with no rip-and-replace of the applications or databases.
- Improved transactional throughput and data ingestion by leveraging GridGain as a distributed IMDB.
- Improved the customer experience or business outcomes by adding HTAP that leverages real-time analytics, streaming analytics and continuous learning.

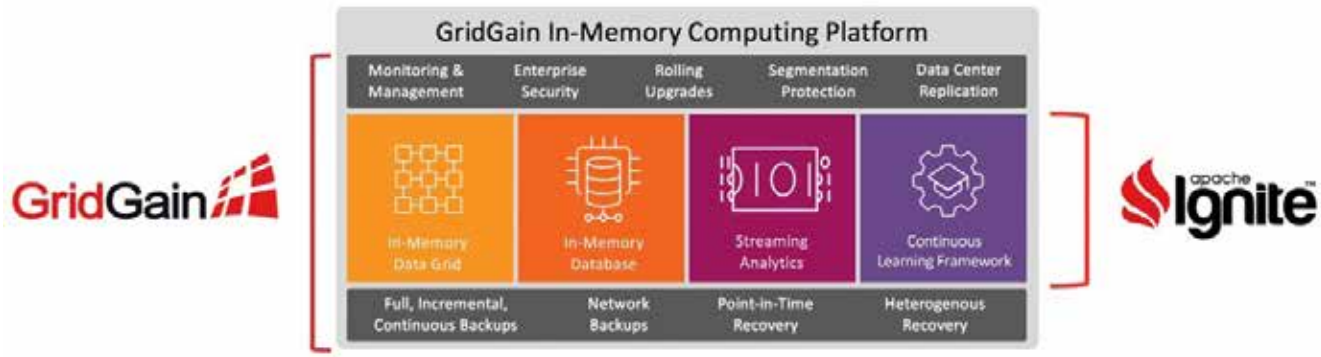

Figure 1. Apache Ignite and the GridGain In-Memory Computing Platform

GridGain customers have been able to create a new shared in-memory data foundation. This single system of record for transactions and analytics enables real-time visibility and action for their business. With each project, they have unlocked more information for use by other applications on a platform with real-time performance at peak loads and always-on availability. As a result, they have been able to develop new projects faster, be more flexible to change, and more responsive in ways that have improved their experiences and business outcomes.

ADDING SPEED AND SCALABILITY TO EXISTING APPLICATIONS WITH AN IMDG

One of the core GridGain capabilities and most common use cases is as an IMDG. GridGain can increase the performance and scalability of existing applications and databases by sliding in-between the application and data layer with no rip-and-replace of the database or application and without major architectural changes.

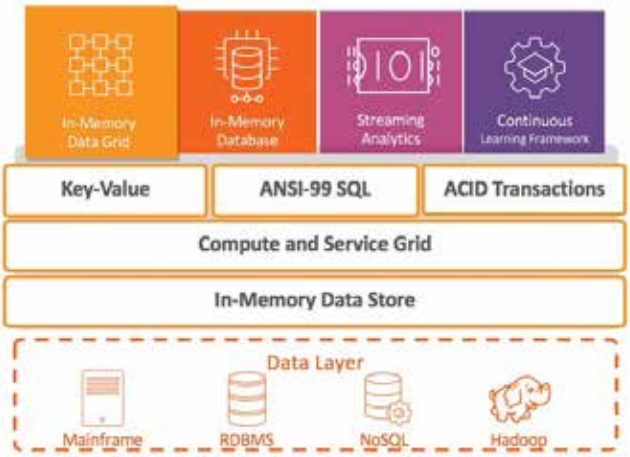

Figure 2. GridGain as an In-Memory Data Grid (IMDG)

This is because GridGain supports ANSI-99 SQL and ACID transactions. GridGain can sit on top of leading RDBMSs including IBM DB2®, Microsoft SQL Server®, MySQL®, Oracle® and Postgres® as well as NoSQL databases such as Apache Cassandra™ and MongoDB®. GridGain generates the application domain model based on the schema definition of the underlying database, loads the data, and then acts as the new data platform for the application. GridGain handles all reads and coordinates transactions with the underlying database in a way that ensures data consistency in the database and GridGain. By utilizing RAM in place of a disk-based database, GridGain lowers latency by orders of magnitude compared to traditional disk-based databases.

STORING DATA FOR HIGH VOLUME, LOW LATENCY TRANSACTIONS AND DATA INGESTION WITH AN IMDB

A GridGain cluster can also be used as a distributed, transactional IMDB to support high volume, low latency transactions and data ingestion, or for low cost storage.

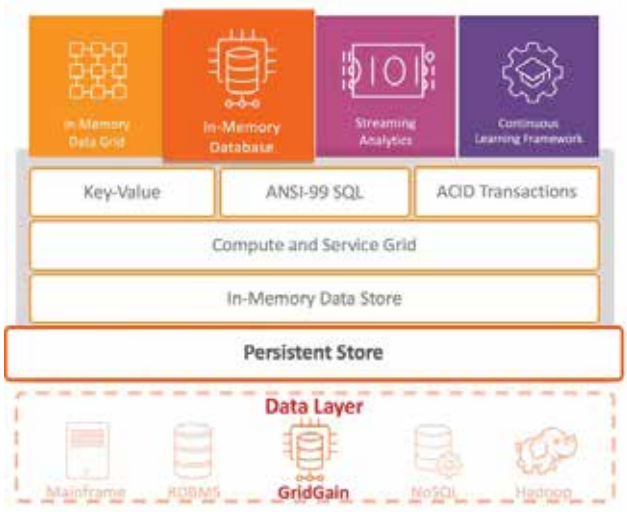

Figure 3. GridGain as an IMDB

The GridGain IMDB combines distributed, horizontally scalable ANSI-99 SQL and ACID transactions with the GridGain Persistent Store. It supports all SQL, DDL and DML commands including SELECT, UPDATE, INSERT, MERGE and DELETE queries and CREATE and DROP table. GridGain parallelizes commands whenever possible, such as distributed SQL joins. It allows for cross-cache joins across the entire cluster, which includes joins between data persisted in third party databases and the GridGain Persistent Store. It also allows companies to put 0-100% of data in RAM for the best combination of performance and cost.

The in-memory distributed SQL capabilities allow developers, administrators and analysts to interact with the GridGain platform using standard SQL commands through JDBC or ODBC or natively developed APIs across other languages as well.

ADDING REAL-TIME ANALYTICS AND HTAP WITH MASSIVELY PARALLEL PROCESSING (MPP)

Once GridGain is put in place, all the data stored in existing databases or in GridGain is now available in memory for any other use. Additional workloads are easily supported by GridGain with unlimited linear horizontal scalability for real-time analytics and HTAP.

GridGain accomplishes this by implementing a general purpose in-memory compute grid for massively parallel processing (MPP). GridGain optimizes overall performance by distributing data across a cluster of nodes and acting as a compute grid that sends the processing to the data. This collocates data and processing across the cluster. Collocation enables parallel, in-memory processing of CPU-intensive or other resource-intensive tasks without having to fetch data over the network.

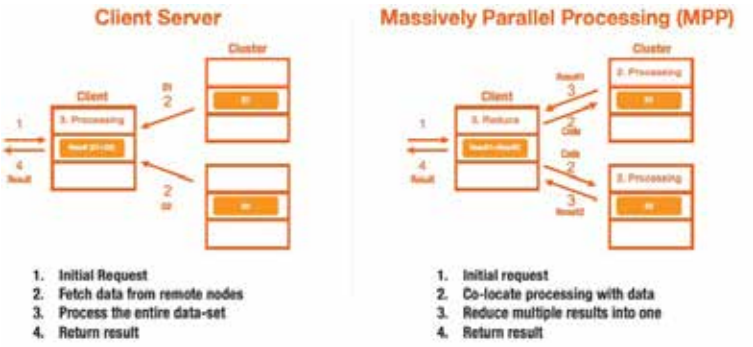

Figure 4. GridGain Compute Grid – Client Server vs Collocated Processing (MPP)

The GridGain Compute Grid is a general purpose framework that developers can use to add their own computations for any combination of transactions, analytics, stream processing, or machine learning. Companies have used GridGain's MPP capabilities for traditional High-Performance Computing (HPC) applications as well as a host of real-time HTAP applications.

GridGain has implemented all its built-in computing on the GridGain Compute Grid, including GridGain distributed SQL as well as the GridGain Continuous Learning Framework for machine and deep learning. Developers can write their own real-time analytics or processing in multiple languages, including Java, .NET and C++, and then deploy their code using the Compute Grid.

Collocation is driven by user-defined data affinity, such as declaring foreign keys in SQL DDL (data definition language) when defining schema. Collocation helps ensure all data needed for processing data on each node is stored locally either as the data master or copy. This helps eliminate the network as a bottleneck by removing the need to move large data sets over the network to applications or analytics.

ADDING DEEPER INSIGHTS AND AUTOMATION WITH STREAMING ANALYTICS AND CONTINUOUS LEARNING

The capabilities GridGain supports are not just limited to real-time analytics that support transactions. GridGain is also used by the largest companies in the world to improve the customer experiences or business outcomes using streaming analytics and machine and deep learning. These companies have been able to incrementally adopt these technologies using GridGain to ingest, process, store and publish streaming data for large-scale, mission critical business applications.

GridGain is used by several of the largest banks in the world for trade processing, settlement and compliance. Telecommunications companies use it to deliver call services over telephone networks and the Internet. Retail and e-commerce vendors rely on it to deliver an improved real-time experience. And leading cloud infrastructure and SaaS vendors use it as the in-memory computing foundation of their offerings. Companies have been able to ingest and process streams with millions of events per second on a moderately-sized cluster.

Figure 5. GridGain for Stream Ingestion, Processing and Analytics

GridGain is integrated and used with major streaming technologies including Apache Camel™, Kafka™, Spark™ and Storm™, Java Message Service (JMS) and MQTT to ingest, process and publish streaming data. Once loaded into the cluster, companies can leverage GridGain's built-in MPP-style libraries for concurrent data processing, including concurrent SQL queries and continuous learning. Clients can then

subscribe to continuous queries which execute and identify important events as streams are processed.

GridGain also provides the broadest in-memory computing integration with Apache Spark. The integration includes native support for Spark DataFrames, a GridGain RDD API for reading in and writing data to GridGain as mutable Spark RDDs, optimized SQL, and an in-memory implementation of HDFS with the GridGain File System (GGFS). The integration allows Spark to:

- Access all the in-memory data in GridGain, not just data streams
- Share data and state across all Spark jobs
- Take advantage of all GridGain's in-memory processing including continuous learning to train models in near real-time to improve outcomes for in-process HTAP applications

GridGain also provides the GridGain Continuous Learning Framework. It enables companies to automate decisions by adding machine and deep learning with real-time performance on petabytes of data. GridGain accomplishes this by running machine and deep learning in RAM and in place on each machine without having to move data over the network.

GridGain provides several standard machine learning algorithms optimized for MPP-style processing including linear and multi-linear regression, k-means clustering, decision trees, k-NN classification and regression. It also includes a multilayer perceptron and TensorFlow integration for deep learning. Developers can develop and deploy their own algorithms across any cluster as well as using the compute grid. The result is continuous learning that can be incrementally retrained at any time against the latest data to improve every decision and outcome.

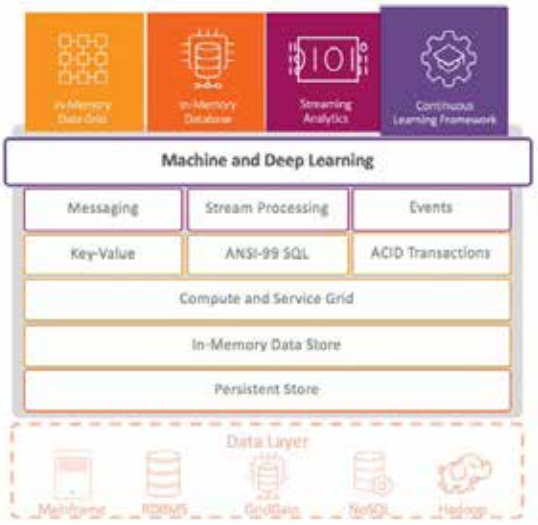

Figure 6. GridGain for Machine and Deep Learning

SUMMARY

Applications and their underlying RDBMSs have been pushed beyond their architectural limits by new business needs, and new software layers. Companies must add speed, scale, agility, and new capabilities to support digital transformation and other business critical initiatives. There are some many options for adding speed and scale to PostgreSQL–and each has its place. But when the speed and scale needs extend beyond what can be addressed at the database layer, the best long term approach is in-memory computing. Not only does it add speed and scale, but it unlocks data, enabling companies to be much more agile.

Contact GridGain Systems

To learn more about how GridGain can help your business, please email our sales team at sales@gridgain.com, call us at +1 (650) 241-2281 (US) or +44 (0)208 610 0666 (Europe), or complete our contact form at www.gridgain.com/contact and we will contact you.

About GridGain Systems

GridGain Systems is revolutionizing real-time data access and processing with the GridGain in-memory computing platform built on Apache® Ignite™. GridGain and Apache Ignite are used by tens of thousands of global enterprises in financial services, fintech, software, e-commerce, retail, online business services, healthcare, telecom and other major sectors, with a client list that includes ING, Raymond James, American Express, Societe Generale, Finastra, IHS Markit, ServiceNow, Marketo, RingCentral, American Airlines, Agilent, and UnitedHealthcare. GridGain delivers unprecedented speed and massive scalability to both legacy and greenfield applications. Deployed on a distributed cluster of commodity servers, GridGain software can reside between the application and data layers (RDBMS, NoSQL and Apache® Hadoop®), requiring no rip-and-replace of the existing databases, or it can be deployed as an in-memory transactional SQL database. GridGain is the most comprehensive in-memory computing platform for high-volume ACID transactions, real-time analytics, web-scale applications, continuous learning and hybrid transactional/analytical processing (HTAP). For more information on GridGain products and services, visit www.gridgain.com.