Whitepaper

Hybrid Transaction and Analytical Processing with NuoDB

Core Tech

Abstract

NuoDB is a distributed SQL database management system that provides a simpler, more cost effective, and ultimately higher-scale approach to hybrid transaction and analytical processing (HTAP). In this paper, we explain the architectural basis for this assertion, and describe an HTAP example to illustrate these unique characteristics.

What Is HTAP And Why Is It Important?

> "Data-driven businesses often need real-time discovery and analysis of transactional data to make meaningful changes immediately."

The combination of simultaneous transactional workloads ("operational") and analytics workloads (business intelligence, Big Data etc.) is a concept that is gaining traction among database vendors and industry analysts. Gartner will be publishing research on the topic in the next 12 months under the "HTAP" moniker, HTAP meaning Hybrid Transaction/Analytical Processing.

HTAP is about spotting trends and being aware of leading indicators in order to take immediate action. For example, an online retailer uses HTAP to find out what products are currently trending as best sellers in the last hour. He sees snow shovels are selling well and reacts by putting a snow shovel promotion on the home page to leverage the trend.

Much of Big Data analytics has focused on information discovery, i.e., essentially using Hadoop for batch-oriented storing and querying of large amounts of data sometimes referred to as "data lakes." However, data-driven businesses often need real-time discovery and analysis of transactional data to make meaningful changes immediately. Real-time discovery and analysis requires repeatable analytics on very recent data. This rules out the traditional approach of offloading transactional data into a separate analytics database for offline analysis.

Unique three-tier architecture

NuoDB has a unique three-tier architecture: an administrative tier, an in-memory caching transactional tier (composed of Transaction Engines, or TEs), and a storage tier (composed of Storage Engines, or SMs). TEs and SMs can run on multiple hosts in one or more datacenters, availability zones, or regions.

By virtue of this architecture, NuoDB has always been capable, as-is, of providing a simpler, more cost effective, and ultimately higher-scale approach to HTAP. In this paper, we'll discuss why this is the case and provide a simple demonstration of these differentiating characteristics.

First, NuoDB supports MVCC (Multiversion Concurrency Control). This means that long-lived analytical queries on recent transactional data won't block ongoing operational workloads. (In addition, NuoDB also supports both "repeatable_read" and "read_committed" transaction isolation levels for further flexibility.)

Second, NuoDB's ability to scale out elastically to accommodate additional workloads means that TE processes can be dynamically added to increase operational throughput, and other TE processes, perhaps specially configured with large memory or compute characteristics, can be added in sole support of analytics workloads as shown in Figure 1 below.

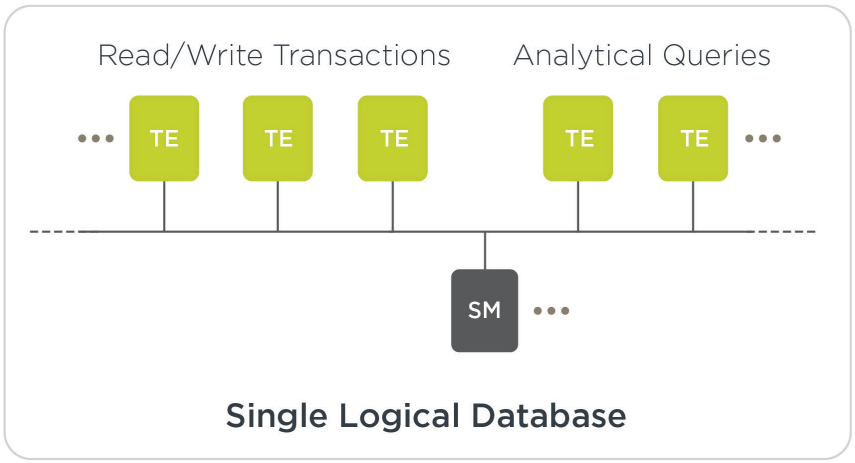

Figure 1: A NuoDB database can run operational and analytics workloads against the same data without impacting the throughput of either workload

In this manner cache pollution is avoided: the in-memory caches for analytics TEs remain "hot" for analytics workloads, while the caches for operational TEs remain "hot" for those workloads. NuoDB supports this workload-specific affinity via an easy-to-configure load balancer.

> "With NuoDB, you can burst out analytics workloads on-demand, then bring those analytics TEs down when they are no longer necessary."

Ideally suited for cloud and burst-out scenarios

These characteristics are especially well suited for cloud deployments involving commodity hardware. NuoDB's unique distributed architecture enables continuous availability and elastic scale in a straightforward manner without changes to the application architecture. You can burst out additional analytics workloads on-demand or on an occasional basis and then bring those analytics TEs down when they are no longer necessary. This is difficult or impossible to do with traditional (client/server) database systems.

SQL/ACID compliant transactions

NuoDB is a fully SQL/ACID compliant database. Operational workloads require ACID semantics. Analytics workloads have an up-to-date and consistent view of the dataset without blocking operational workloads via MVCC-mediated access to the latest data. The result is that these two kinds of SQL workloads—short running write-intensive transactions and long running read-intensive transactions—can coexist efficiently.

Ideal HTAP configuration performance

With this background in mind, let us now explore how the ideal HTAP configuration should perform. What you would want is a system that delivers uniform operational throughput even as an analytics workload is added to the mix.

The ideal HTAP performance characteristics are depicted in the chart below. The left-side Y-axis represents total operational throughput and the right-side Y-axis represents total analytics throughput. The X-axis represents the mix of analytical versus operational transactions.

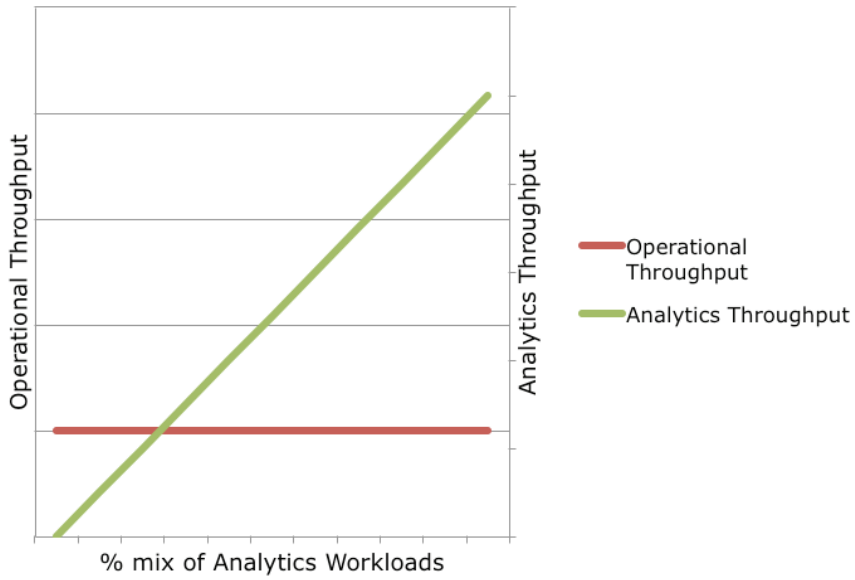

Figure 2: Ideal HTAP system behavior

In this ideal scenario, the operational workload remains unaffected as more and more analytics workloads are added.

NuoDB's architecture will adhere closely to this ideal performance envelope. However, with a traditional SQL database, you would expect that the workloads would interfere with each other, with subsequent degradation in performance. Consequently, to support HTAP, you would have to over-provision a traditional SQL database with hardware capable of handling the most extreme HTAP case you expect to encounter.

Why Traditional DBMS Fail To Deliver HTAP

With the preceding assumptions in mind, let us now dig in deeper into why traditional client/server database management systems (DBMSs) were not designed for demanding HTAP use cases. Their limitations to adequately support HTAP fall broadly into these categories:

- Lock contention
- Inability to scale (memory, I/O)

Lock contention

Conventional lock-based DBMSs have trouble executing long-running transactions because readers block writers and writers block readers. For example, if a reader needs data from a specific column in the database, it locks out the writer while it retrieves that data and vice-versa.

Analytical processing must simultaneously read large portions of the database thereby increasing the chances for a lock conflict. The natural consequences of lock conflicts are that you will experience noticeable performance drops when you run analytics workloads on the same DBMS as your operational workloads.

A solution to this problem is to use a database that supports multi-version concurrency control (MVCC). An MVCC-based DBMS supports snapshot isolation allowing a long-running reader (such as an analytics query) to operate on a historical snapshot of the database while newer versions of records are independently created or mutated by OLTP-style INSERT and UPDATE loads.

Inability to scale

Even with MVCC, HTAP scenarios involve analytics workloads that operate in parallel, leading to yet another gating factor: the inability to scale. As the intensity of the analytics workloads increases, pressure on CPU, memory, network, and I/O resources will increase and begin to affect throughput for all workloads, which is unacceptable for critical operational workload-based business processes.

> “Optimizing an HTAP system is not just about properly configuring and sizing the system for different workloads.”

To compound the challenge, optimizing an HTAP system is not just about properly configuring and sizing the system for different workloads. It also requires that you understand that things like cache line, paging, prefetch, and disk scan patterns are different for operational and analytics workloads. The traditional HTAP system cannot optimize for both at the same time, so thrashing occurs as the two patterns compete.

One response to these kinds of resource pressures, and in order to protect operational workload performance in an HTAP scenario, is to run a traditional RDBMS on a very high-performance server configured with a large number of processing cores, large memory, high-capability network and I/O subsystem. This “scale-up” approach may provide additional headroom, but is very expensive and ultimately limited to the capabilities of the largest server available to you.

In addition, scaling an HTAP system based on a traditional DBMS in cloud-based environments is yet another limiting factor, because these environments tend to be based on commodity hardware with fewer numbers of cores, less memory, and less-capable network and I/O. Inherently, the cloud is the antithesis of scale up; rather, it’s about elasticity—scaling out and back in—providing on-demand, commodity resources at the lowest total cost of ownership.

An HTAP Example

To demonstrate the HTAP capabilities of NuoDB, the following three workloads were designed and tested on the AWS cloud using NuoDB Swifts Release 2.1, to simulate a real-world mix of operational and analytics workloads:

- *Load workload:* create a database on which operational and analytics queries could operate
- *Operational workloads:* repeatedly query the database and update selected records. The database traffic for this workload is directed to a TE dedicated to this type of workload
- *Analytics workloads:* query the database for recently updated records. The database traffic for this workload is directed to two TEs dedicated to this workload.

"The HTAP features inherent in the NuoDB architecture and emphasized in their latest product release have the potential to radically improve and enhance operational BI and analytics."

Dr. Barry Devlin,
Founder,
9sight Consulting

The "Load" workload is run once, resulting in a database containing a configurable number of rows in a single table. Multiple instances of the "Operational" workload are then started in parallel on a single host and their performance measured as they repeatedly query and update the table. A short while later, multiple instances of the "Analytics" workload are stagger-started in parallel on two hosts, and their performance monitored as they query for records recently updated by the operational workload. The analytics workloads are then halted while the operational workloads are allowed to continue, to examine how the operational workloads react and recover.

These workloads were implemented as a single Java application that can execute any of the three workloads (by specifying a command-line parameter).

NuoDB configuration in AWS

A NuoDB Domain is configured in AWS, with 5 EC2 instances configured as follows:

- An "operational" host on which 5 instances of the operational workload are run in parallel, with a co-located NuoDB TE dedicated to servicing operational queries.
- 2 "analytics" hosts, each running up to 5 instances of the analytics workload, and each with a co-located NuoDB TE dedicated to servicing analytics queries.
- 2 hosts each running a NuoDB SM to match a typical redundant configuration.

The EC2 "m3.xlarge" instance type was used for all hosts, which offers 4 vCPUs, 15GB memory, and 2x40GB SSD. See the EC2 documentation for details.

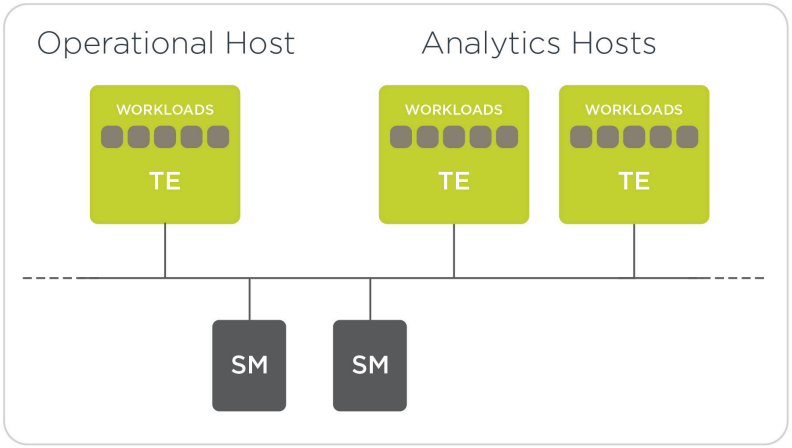

Figure 3: NuoDB HTAP test configuration on Amazon AWS

This configuration, seen in Figure 3, exploits NuoDB's distributed architecture by isolating the two kinds of workloads from each other. The operational workloads can monopolize the CPU and memory of its associated TE without being concerned about affecting the performance of the analytics workloads, and vice versa, and the cache for each TE remains "hot" for each kind of workload, avoiding "cache pollution" that might occur if both kinds of workloads had to share a single cache.

Database

A NuoDB database is created using the Automation Console "Multi Host" template. This can be accomplished via command line, REST API, or via the "Create Managed Database" dialog in the Automation Console.

NuoDB 2.1 supports affinity-based load balancing, by which application requests are directed to specific sets of TEs. In this way, operational workloads can be directed to TEs configured for operational workloads while analytics workloads are directed to TEs that may be optimized for these kinds of workloads.

For this test, 3 TEs and 2 SMs are made available for this database per its "Multi Host" template. (See the appendix for the details)

Data model

The database contains a single table, "`STATE_DATA`", which represents simulated update transactions that occur in any of the US states. The table includes five columns, and indexes are created against two of the columns. The simulated transactions in each row record are:

- A pseudo-randomly selected NAME of a US State in which the simulated transaction occurred
- A pseudo-random ID for the transaction
- A creation timestamp TS
- A last-updated timestamp TSU
- A pseudo-randomly generated numeric VALUE

(See the appendix for the details)

Load workload

When performing the initial table load, the application, running in "Load" mode, inserts a pseudo-randomly selected US State name into the NAME column, pseudo-randomly generated integers limited to a specified range into the ID and VALUE columns, and the current date/time into the TS and TSU columns. The number of rows to load is configurable.
(See the appendix for details)

Operational workload

When running the operational workload, the code first performs a SELECT with a pseudo-randomly generated ID value specified in the WHERE clause (See the appendix for details). If the result set is not empty, an UPDATE is performed against all rows sharing this ID, setting the value in the TSU column to the current date/time. Consequently, values in the TSU column are constantly being updated, simulating a transactional workload.

Analytics workload

When running in the analytics mode, the code selects records in which the TSU column contains a value that's been updated within the last five minutes; this is expected to yield tens of thousands of results in a database of a million or more rows:

```
select * from state_data where tsu between ? and ?
```

where the query parameters are replaced at runtime to select for records that have been updated in the last five minutes.

Logging results

When in operational or analytics mode, the workload application emits a log statement on a configurable periodic basis. The statements look something like this for the operational workload:

```
OPERATIONAL          1414280660719          537.4
OPERATIONAL          1414280661719          516.8
OPERATIONAL          1414280662719          501.9
OPERATIONAL          1414280663719          490.8
OPERATIONAL          1414280664719          475.6
```

Or this, for the analytics workload:

```
ANALYTICS            1413679858461        78462.0
ANALYTICS            1413679859461       118087.5
ANALYTICS            1413679860461       157964.3
ANALYTICS            1413679861461       158312.3
```

The first column indicates the type of workload to facilitate post-processing. The middle column is a timestamp, and the third column indicates the number of rows processed:

- *Operational*: number of rows updated in the last 1 second interval
- *Analytics*: number of rows identified in the last n second interval as having been updated in the last 5 minutes. For these tests, a row is emitted every 5 seconds, and this data normalized to 1 second for the results reported below.

Test procedure

These steps are taken to run the test:

1. The workload application is run in "Load" mode to load the desired number of rows. For this test, 5 million rows are loaded. This step is run once on the host configured to service operational workloads.
2. On the operational host, 5 instances of the workload application in "Operational" mode are started in parallel and allowed to run for several minutes.
3. On the two analytics hosts, 10 instances of the workload application in "Analytics" mode are stagger-started as follows:
   a. On one of the two analytics hosts, an instance of the analytics workload is started.
   b. A short while later (~30 seconds), a second instance of the analytics workload is started.
   c. This pattern is repeated until 5 analytics workloads are running on one analytics host.
   d. A short while later, a first instance of the analytics workload is started on the second anayltics host.
   e. More analytics workloads are started, after short delays, until 5 are running on the second analytics host.
4. The full set of workloads – 5 operational and 10 analytics – are run in parallel for a while.
5. The analytic workloads are halted, and the operational workloads allowed to run on their own for a while longer.

Test results

The graph below shows the results of the test run. The Y-axis on the left side of the graph corresponds to the operational workload (rows updated in the last 1 second interval), and the Y-axis on the right side corresponds to the analytics workload (number of rows identified per second as having been updated in the last 5 minutes by the operational workloads). The X-axis plots the number of simultaneous analytics workloads in operation.

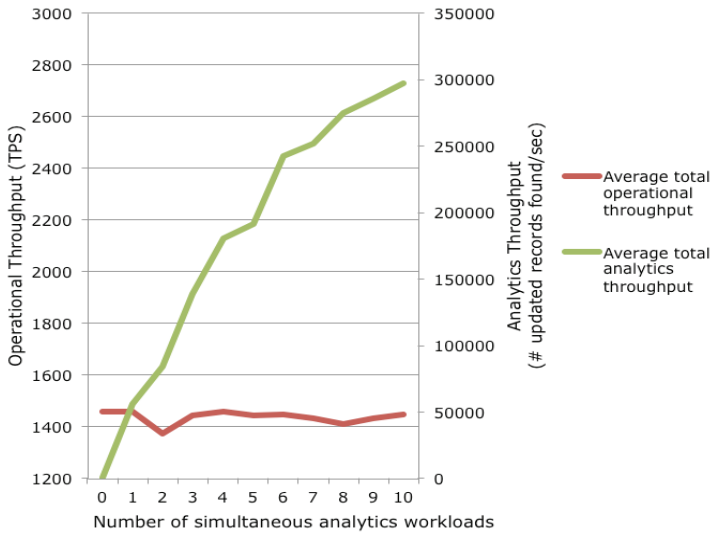

Figure 4: Results of HTAP test on NuoDB

"We are seeing growing interest in databases that combine transactional and analytics capabilities to deliver operational intelligence. With its latest release NuoDB once again demonstrates that it is at the forefront of innovation being delivered with NewSQL relational databases."

Matt Aslett,
Research Director,
451 Research

Discussion of results

Observations and comments about these results:

1. The operational throughput remains relatively steady throughout the test, at about 290 updates per second for each of the operational workloads, for a total throughput approximately of ~1,450 TPS.
2. When all 10 analytics workloads are running, the throughput of each is in the range of 80,000 to 170,000 updates found per 5 second period, for a total throughput of about 1,400,000 updates found per 5 second period, normalized to about 280,000 per 1 second period in the graph.

The CPU loading for the various hosts was monitored throughout the test. When all 10 analytics workloads are running, the two hosts running the analytics workloads and TEs are at about 90% CPU. The CPUs on the operational workload and two SM hosts are in the 10% - 25% range.

The key findings are:

- The 5 operational workloads are unaffected by the analytics workload, even as the number of those analytics workloads is increased to 10 parallel instances.
- The analytics workloads show a steady ramp in total throughput as the number of parallel instances is increased to a total of 10. The resulting total throughput indicates a linear horizontal scale-out model.

These results support the premise that NuoDB is architecturally suited to supporting scale-out of diverse workloads.

Conclusion

This simple example demonstrates the following architectural characteristics of NuoDB:

- *MVCC support:* analytical queries on recent transactional data won't block ongoing OLTP workloads.
- *Performance scale-out:* dynamically scale-out horizontally to accommodate additional simultaneous diverse workloads. Burst out for short-lived analytics workloads and scale back down later when no longer needed.
- *Isolated distributed caches:* TEs avoid cache pollution since caches stay "hot" for specific assigned workloads.

Additional information on NuoDB and HTAP can be found at: **www.nuodb.com/HTAP**

It is this combination of characteristics that make NuoDB uniquely and especially well suited for HTAP. IT Managers can easily and effectively use NuoDB's advanced automation capabilities to configure applications to take advantage of these architectural advances.

The NuoDB Community Edition can be downloaded for free if you want to run this test yourself: **www.nuodb.com/download**.

Appendix

Configuring the NuoDB domain and database for HTAP

NuoDB's affinity balancer, introduced in version 2.1, enables the tagging of hosts so that each kind of workload can direct its database traffic to the appropriate TE.

For this test, NuoDB was installed onto 5 hosts in EC2, and the "default.properties" files on each were configured as follows:

- Operational host: hostTags = TE = 1, LBTag = Operational
- Analytics host #1: hostTags = TE = 1, LBTag = Analytics
- Analytics host #2: hostTags = TE = 1, LBTag = Analytics
- SM host #1: hostTags = SM = 1
- SM host #2: hostTags = SM = 1

In the database's Tags section in the Automation Console, the Template variables SM_HOST_TAG and TE_HOST_TAG are set to "SM" and "TE" respectively.

For each workload, the JDBC connection string is customized for the appropriate workload as follows

```
jdbc:com:nuodb://host:port/database?LBTag=Operational
```

for operational workloads, and:

```
jdbc:com:nuodb://host:port/database?LBTag=Analytics
```

for analytics workloads.

When the workload application initially requests a connection to the database, a NuoDB Broker handles the request. The Broker determines which TE should be given the request, taking into account the LBTag. The initial request is then routed to the selected TE connection; subsequent requests go directly to that TE.

HTAP data model

The following NuoDB DDL defines the table and indexes:

```
CREATE TABLE IF NOT EXISTS "WORKLOAD"."STATE_DATA" (
"ID" bigint NOT NULL,
"TS" timestamp NOT NULL,
"NAME" varchar(50) NOT NULL,
"VALUE" bigint NOT NULL,
"TSU" timestamp NOT NULL
      );

CREATE INDEX "ID_IDX" ON "WORKLOAD"."STATE_DATA" ( "ID" );

CREATE INDEX "TS_IDX" ON "WORKLOAD"."STATE_DATA" ( "TSU" );
```

You can submit this DDL in any of several ways:

- The NuoDB nuosql utility
- The NuoDB SQL Explorer interactive SQL utility
- Any compatible 3rd-party SQL utility

"ID" column details

Each row in the "STATE_DATA" table includes an ID column that, while not unique, exhibits very limited redundancy. An index is defined on this column. The value inserted into the ID column when a record is first added to the table depends on the planned size of the table. That is, a 5 million-row table will have a different set of IDs than a 40 million-row table.

The planned table size is used as a seed into an instance of a java.util.Random object such that a pseudo-random number between zero and the number of rows is generated. This value is then multiplied by 1,000 and added to 100 million to yield the ID value and a sparsely populated index. This code repeats for each row inserted into the table.

```
Random rnd_id = new Random();
int id = (rnd_id.nextInt(recordCount) * 1000) + 1000000000;
// recordCount is the desired size of the table
```

When running the operational client, the same code is used to generate an ID that is used in a SELECT statement WHERE clause to return zero or more rows with this ID. If the result set is not empty (that is, resultSet.next() returns true) then the ID value is used in an UPDATE statement WHERE clause and a timestamp column (TSU) in the record is set to the current datetime.

The analytic client code does not depend on the ID. Its query depends on the value in the TSU column (on which there is also an index).

About NuoDB

NuoDB was launched in 2010 by industry-renowned database architect Jim Starkey and accomplished software CEO Barry Morris to deliver a webscale distributed database management system that is specifically designed for the cloud and the modern datacenter. Used by thousands of developers worldwide, NuoDB's customers include automotive after-market giant AutoZone, Europe's second largest ISV Dassault Systèmes, Zombie Studios and many other innovative organizations. NuoDB is cited for the second consecutive year on the industry's leading operational database management systems analysis. NuoDB is based in Cambridge, MA. For further information visit: **www.nuodb.com**.

215 First Street
Cambridge, MA 02142
+1 (617) 500-0001
www.nuodb.com

Version 1 – 11/5/2014