CH2++: New HOAP for Benchmarking JSON Data Analytics

M.J. Carey, V. Sarathy, D. Nagy, B-C. Wang, K. Murthy, M. Muralikrishna, P. Gupta, and T. Westmann

Couchbase, Inc., San Jose, CA, USA
mike.carey@couchbase.com

Abstract. Database systems with hybrid data management support, referred to as HTAP or HOAP architectures, have gained popularity for performing near-real-time data analytics. Such systems first emerged in the relational world, and the CH-benCHmark (CH) was proposed in 2011 to evaluate such relational systems. A decade later, with NoSQL database systems gaining traction for new application development, we proposed a first document-oriented variant of CH – called CH2 – that was aimed at evaluating hybrid data platforms in the document database space. Here, we introduce CH2++, a major revision of CH2 that is significantly more suitable for evaluating systems that support JSON analytics.
Like CH and CH2, this new CH2++ benchmark borrows from and extends both TPC-C and TPC-H. Differences from CH2 include a much more document-oriented schema, the inclusion of additional fields to exercise analytical systems' column-oriented features, and a corresponding JSON "do over" of the CH2 queries. This paper also includes preliminary performance isolation and scaling results from running CH2++ against Capella Operational and Capella Columnar, which are Couchbase's DBaaS cloud service offerings. The results highlight the value of CH2++ for evaluating JSON-targeted HOAP and HTAP platforms.

Keywords: Benchmarks · NoSQL · JSON · HTAP · HOAP.

1 Introduction

In today's world, organizations are becoming ever more reliant on real-time information and its analysis to steer and optimize their operations. Historically, operational (OLTP) and analytical (OLAP) processing were two separate activities, each running on their own separate infrastructures. Periodic ETL processes served to bridge these worlds in the overall IT architecture of a typical enterprise [22]. Today, database management systems with hybrid data management support – known as HTAP (Hybrid Transactional/Analytical Processing [33]) or HOAP (Hybrid Operational/Analytical Processing [1]) support – have appeared on the scene and gotten traction in both industry and research in order to address the pressing needs for timely analytics. Born in the relational world, these

hybrid platforms are often associated with other concurrent high-end server technology trends. Columnar storage and main memory data management are two technologies that are often assumed to be part of that picture.

While relational databases still dominate the IT landscape, today's applications demand support for millions of interactions with end-users via the Web and mobile devices. Traditional relational database systems were built to target thousands of users. Designed for strict consistency and data control, relational database systems can fall short of the agility, flexibility, and scalability demands of today's new applications. This led to the emergence of a category of data management systems known as NoSQL systems [30]; our focus here will be on NoSQL's subcategory of document databases. Examples of such systems include Couchbase Server [3] and MongoDB [16]. NoSQL systems aim to provide incremental and horizontal scalability on clusters of computers as well as to eliminate the mismatch between an application's view of data and its persisted form, thus enabling players, ranging from application developers to DBAs and data analysts, to work with their data in its natural form.

Given these developments, one might wonder if the world of NoSQL is HOAP-less[1], particularly the world of document databases. The answer is no, as the need to combine operational and analytical capabilities for up-to-the-minute analytics exists there as well. NoSQL vendors are starting to provide enterprise customers with HOAP and include HOAP-ful messaging in their marketing materials. One such vendor, one whose document data management capabilities we will benchmark here, is Couchbase; the Couchbase Server platform first introduced HOAP with its addition of Couchbase Analytics [15]. Today, Couchbase's Capella DBaaS cloud offering provides HOAP via its Operational and Columnar services. However, our benchmark is not specific to Couchbase. It can be applied to HOAP-ful configurations of other NoSQL databases as well; that, in fact, is the very point of this paper. Here we propose a benchmark for evaluating HOAPfulness in the world of document data management, the world where JSON is the dominant data model and where the elimination of the ETL requires analytical query support for JSON.

In the relational world, the problem of evaluating database system performance under hybrid OLTP/OLAP workloads has attracted attention. Perhaps the most prominent example is the mixed workload CH-benCHmark [10] (CH). CH was proposed by a gathering of database query processing and performance experts and has since been used to evaluate the performance of new HTAP systems [29]. The same is not yet the case in the NoSQL world; a truly successful benchmark to assess HOAP for scalable NoSQL systems has yet to appear. We offered a first NoSQL HOAP benchmark called CH2 [7] based on extending and improving CH in several important ways, but it has not enjoyed significant uptake outside of Couchbase. In this paper we propose take two – namely CH2++, a derivative of CH2 with a much more appropriate JSON schema.

[1] We prefer the term HOAP over HTAP for NoSQL; the term HTAP seems more tied to strict ACID transactions and main-memory technology presumptions.

The rest of this paper is organized as follows: Section 2 surveys work on HTAP and on prior SQL and NoSQL benchmarks. Section 3 describes CH2++, our proposed HOAP-for-NoSQL benchmark, explaining how it differs from CH and CH2. As a HOAPful document platform example, Section 4 provides an overview of Couchbase's Capella cloud DBaaS offering and its approach to HOAP. To demonstrate CH2++'s potential to be an effective hybrid workload benchmark for NoSQL, Section 5 presents a set of results from running CH2++ on Capella under different service configurations. Section 6 summarizes the CH2++ proposal and invites others to take CH2++ for a test drive of their own.

2 Related Work

We first review related work on HTAP/HOAP and database benchmarks.

2.1 HTAP (HOAP)

The relational database world has seen the emergence of HTAP (a.k.a. HOPE) capabilities in a number of commercial systems as well as growing research interest related to HTAP. Notable HTAP offerings include HyPer [17], born in research but now owned and used by Salesforce in Tableau, and SAP-HANA [21]. Other relational HTAP offerings include IBM's DB2 BLU [28], Oracle's dual-engine main-memory database [18], and the real-time analytical processing capabilities in Microsoft's SQL Server [19]. As an example on the research side, a relatively recent paper introduced and explored the concept of adaptive HTAP and how to manage the CPU core and memory resources of a powerful (scale-up) many-core NUMA server for handling a mixed main-memory workload [29].

HTAP in the relational world has focused mainly on in-memory scenarios involving modestly sized operational databases. With the availability of multi-core servers with very large main memories, and compression from columnar storage, it is now possible for main memory to hold much or all of an enterprises' operational data. This is why most current HTAP offerings have focused on main-memory database technology. And, as might be expected, their focus has been on single-server architectures, i.e., on scale-up rather than scale-out.

Bringing HOAP to the world of document data management involves different problems that need different solutions. To scale document databases while providing HOAP, the focus needs to be on Big Data, i.e. large volumes of flexible, schemaless data. Also, NoSQL systems and their applications tend to have different transactional consistency requirements [30]. Data timeliness is important in the NoSQL world, but there is less of a need to focus on reducing or eliminating ACID transaction interference and a greater need to focus on providing strong performance isolation at the level of a cluster's physical resources.

2.2 Benchmarks

There are many benchmarks for evaluating the performance of relational database systems in various application scenarios [14]. The most notable benchmarks are

the Transaction Processing Council (TPC) TPC-x benchmarks, including TPC-C [27] for transaction processing, as well as TPC-H [25] and TPC-DS [26] for decision support and analytics. There have also been various benchmarks proposed and used in the NoSQL world, including YCSB [11] for key-value stores, BigFUN [23] for Big Data management platform performance, MongoDB's adaptation of TPC-C to evaluate NoSQL transactional performance [16],and a similar NoSQL adaptation [24] of TPC-H to evaluate Big Data analytics performance, to name a sample of the NoSQL and Big Data benchmarks.

A recent survey examined HTAP databases and benchmarks [20]. An especially notable HTAP benchmark is the CH benCHmark for mixed workloads [10]. CH-benCHmark (CH) came from a Dagstuhl workshop attended by database query processing and performance experts from both industry and academia. CH combined the schemas and database operations of TPC-C and TPC-H to bridge the divide between the established single-workload benchmark suites of TPC-C for OLTP and TPC-H for OLAP, thereby providing a foundation for mixed-workload performance evaluation. The original CH paper included results from running CH on PostgreSQL with all data in memory and a read-committed isolation level. CH gained traction for HTAP evaluation, including being used to assess the performance of an adaptive HTAP system and its scheduling ideas [29]. Several other HTAP benchmarks have appeared subsequently, including HTAP-Bench [9] and HyBench [34]. HTAPBench is a descendent of CH; it proposed setting a fixed transaction performance target and increasing the analytical query workload until measurable transaction interference is observed relative to the target. HyBench is based on a different application scenario and it also added interactive transactions (OLXP) to the workload mix. Both of those benchmarks were still focused on the evaluation of relational HTAP systems.

To our knowledge, our CH2 [7] extension of CH, discussed in more detail next, was the first mixed workload benchmark intended to evaluate HOAP for scalable NoSQL (JSON-based) systems. An earlier exploratory step was [32], where performance isolation in Couchbase Server was studied for a simple scenario with lightly nested TPC-C data and a mix of NewOrder transactions and a stream of join/group-by/top-K analytical queries. Our subsequent CH2 effort was suggested as future work at the end of that earlier article. As was the case for CH2, the aim of the CH2++ benchmark in this paper is to evaluate HOAPful platforms and services in the document database space.

3 CH2++ Benchmark Design

When we started the CH2 effort that preceded this work, our goal was to explore HOAP support on NoSQL platforms, including (1) their delivery of performance isolation for mixed OLTP and OLAP workloads, and (2) their query performance for OLAP-style queries. These points were of particular interest because most NoSQL systems were designed to scale horizontally and their query engines have been OLTP-oriented, so NoSQL systems have generally aimed to support high-concurrency/low-latency operational workloads rather than richer analytics.

As discussed in [7], while reviewing existing relational HTAP benchmarks, we learned of the CH benchmark [10]. We were happy to find that its approach was to adopt and merge the data, operations, and queries from the well-regarded TPC-C and TPC-H benchmarks. Having had similar thoughts, we decided for CH2 to adapt CH's relational schema to the JSON world by re-modeling orders as objects with embedded line item arrays (rather than as two separate 1NF tables), as was also done in [16] and [24]. While doing that, we also encountered and fixed a number of problems with the original CH design that required more extensive changes to CH's data and query predicate generation. In the remainder of this section we describe CH2++, our recent re-work of CH2, including its reason(s) for being and its differences from CH2.

3.1 CH2++ Database Schema

Figure 1 summarizes the schema of the CH2++ benchmark, including its eleven collections and their relationships. We borrow the optionally typed collection DDL from Apache AsterixDB [5] to convey the structure of the JSON objects in each collection, with solid black links to indicate the N:1 relationships between them. The numerical annotations on the links show the number of child objects per parent (e.g., 10 districts per warehouse, 3000 customers per district). Like CH and CH2, the CH2++ database is scalable by varying the number of warehouses (W). Sizes for three of the collections, supplier (10K objects), region (5 objects), and nation (62 objects), are fixed instead of scaling with W. CH and CH2 borrowed those collections from TPC-H; the other eight are either taken from TPC-C or from both (using TPC-C's naming conventions).

To relate objects in the TPC-H inspired collections to objects in TPC-C, CH2++ does what CH (and then CH2) did: An entry in stock is associated with its supplier through the functional relationship `stock.s_i_id * stock.s_w_id mod 10,000 = supplier.su_suppkey`. Customer nations are identified by the first character of `customer.c_state`. In TPC-C this character can take any of 62 different values (upper- and lower-case letters and numbers), so CH chose 62 nations (vs. 25 in TPC-H) to populate the nation collection. The values for `nation.n_nationkey` are chosen so their associated ASCII values are letters or numbers. The region collection then contains five regions for these nations. Linkages between these collections are modeled via the foreign key fields `nation.n_regionkey` and `supplier.su_nationkey`. These functional linkages (stitching) are denoted by the dotted links and formulae shown in Figure 1.

As mentioned above, the design goal for CH2++ was to create an effective benchmark for HOAP workloads involving JSON analytics. CH was a modest relational melding of TPC-C and TPC-H, and the original CH2 was a minimal JSON-ification of CH that simply nested orderlines within orders. In contrast, CH2++ is a modern-day update of CH2 that we arrived at by asking the question "what might TPC-C and TPC-H have looked like, schema-wise, if they were designed today for document-based applications?" We answered this question by following current best practices for JSON database design [4]. As a result, CH2++'s schema is much more JSON-ic in nature than that of CH2:

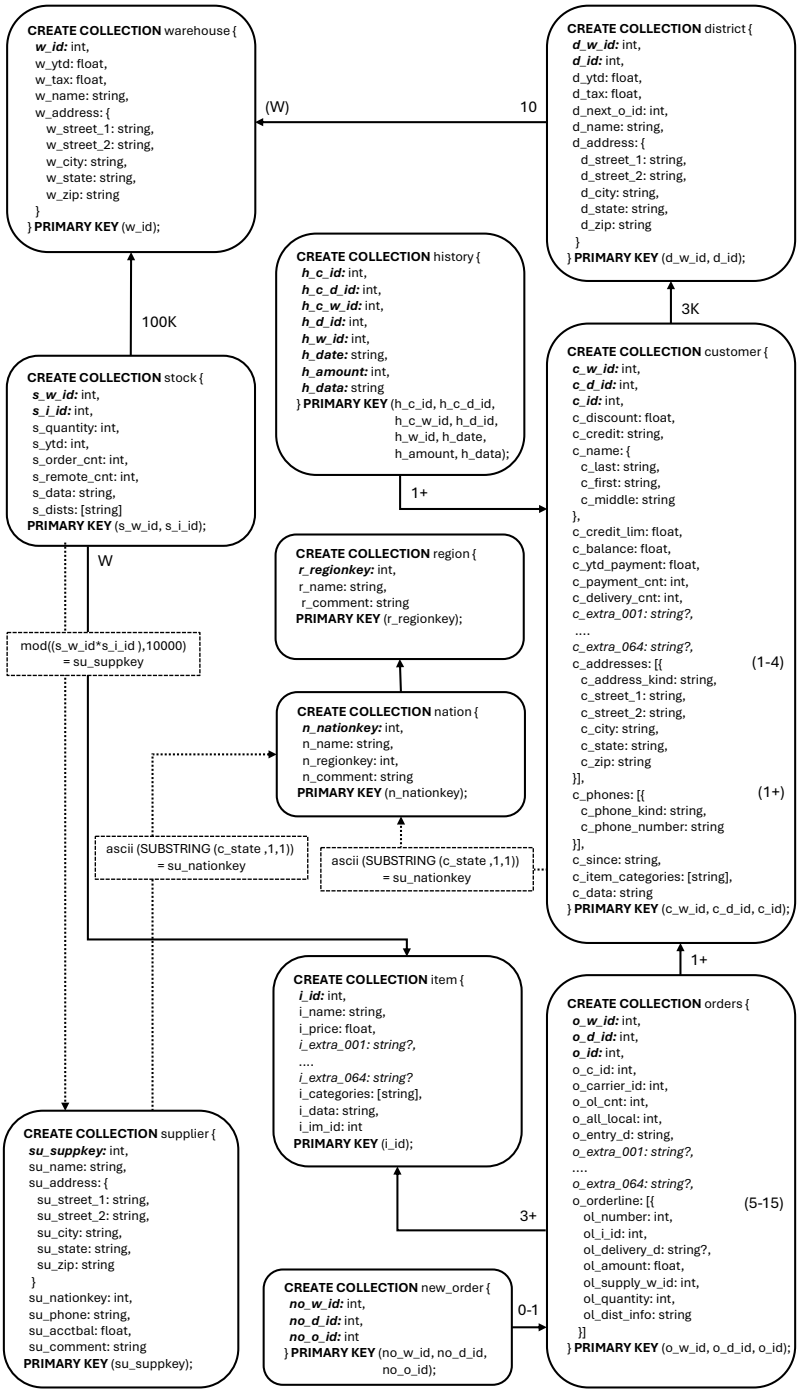


Fig. 1. CH2++ Schema

- Addresses and names are now modeled as nested objects wherever they appear in any of the collections.
- Customers now have an array of 1-4 addresses (shipping, home, work, and/or billing) with a shipping address being mandatory and the others each occurring with 1/3 probability.
- Customers now have an array of 1-4 phones (contact, home, work, mobile) with a contact phone being mandatory and the others each occurring with 1/3 probability.
- Customers now have an additional array of 0-15 category names (drawn from 128 possibilities) to capture the kinds of items of particular interest to them.
- Items now have an additional array of 1-3 categories (drawn from the 128 possibilities) to indicate what kind(s) of item they are.
- Stock entries now have a 10-element array of inventory levels instead of having ten separate, numerically named fields.
- Several of the collections (customer, item, orders) now have 64 extra fields to simulate the modern-day reality that typical data objects in 2025 can be much wider than those in the 1990s (when TPC-C and TPC-H were born).

Table 1 list the CH2++ collections and gives an example of their scaling by showing cardinalities for a 1000-warehouse benchmark instance. As in CH2, orders have a nested orderline array with an average of 10 line items. The line in the table separates the 8 borrowed TPC-C collections from the 3 borrowed TPC-H collections.

Collection	**Cardinality (W=1000)**
warehouse	1,000
district	10,000
history	30,000,000
neworder	9,000,000
stock	100,000,000
customer	30,000,000
orders (orderline)	30,000,000 (300,000,000)
item	100,000
supplier	10,000
nation	62
region	5

Table 1. CH2++ Collections and Cardinalities for 1,000 Warehouses

3.2 CH2++ Database Data

As mentioned above, when initially generating the database for CH2, we found that the CH (and TPC-C) data generators had issues in generating data suitable for the analytical queries. The biggest problem can be described as the TPC-C "big bang" – all date fields in the initial CH database had the current date (i.e. the date of a benchmark run) as their value, so there were no useful date ranges in the data for analytical queries. (TPC-H's queries operate on a 7-year history.) Thus, while CH's queries were based in principle on TPC-H, many predicates returned nothing, everything, or a run-date-dependent result.

To address this in CH2 [7], we created a new data generator using the TPC-C data generator from CMU's py-tpcc benchmarking package to (i) generate orders with nested items and (ii) carefully generate values for fields used in the predicates of the analytical queries. For (ii) we introduced a RUN_DATE parameter to drive data generation; RUN_DATE led to derivative parameters START_DATE and END_DATE that are similar to TPC-H's date range. RUN_DATE controls the historical characteristics of the data by determining when the benchmark's past should end and when its operational life should begin, i.e., START_DATE = RUN_DATE - 7 years and END_DATE = RUN_DATE - 1 day. We also adjusted the dates in the data generator based on TPC-H, added the supplier, nation, and region collections (a change relative to py-tpcc), and adjusted the dates and ranges in the analytical queries to conform to TPC-H's predicate selectivities and business semantics [7].

In addition to these changes, to transition from CH2 to the now richer CH2++ schema, we modified our CH2 generator to create nested objects for the nested names and addresses, to use array-based modeling of customer addresses and phones (with the mandatory shipping address and mandatory contact phone carrying the address and phone values that CH2 had before the additional kinds of addresses and phones were added), to add category arrays to customers and items, and to add 64 extra fields to customers, orders, and items. (Note: Although 64 is the setting used in this paper, it is a tuneable CH2++ parameter.) None of the extra fields are actually accessed in the benchmark's queries, as their role is to test an analytical system's ability to avoid touching unused data.

3.3 CH2++ Transactional Operations

The operational workload of TPC-C models transactions for a typical order processing scenario. The transactions are a mix of five read-only and update-intensive business transactions: NewOrder, Payment, OrderStatus, Delivery, and StockLevel. CH2++, like CH and CH2, uses this mix as its operational workload. Like its predecessors, CH2++'s operational performance reporting focuses on NewOrder, a transaction that enters a new order (containing multiple orderlines) into the database. NewOrder touches most of the TPC-C tables, and consists of both read-only queries and updates against them. Details of the five TPC-C transactions as well as the prescribed mix can be found in [27]. One change that CH2 and CH2++ made to py-tpcc's TPC-C implementation is to chop Delivery, which does delivery processing for ten orders, into ten per-order transactions rather than grouping them as one transaction. This is more in line with what the TPC-C specification prescribes (Section 2.7 in [27]) and follows best practices for selecting transaction boundaries [31]. Also, py-tpcc deviates from the TPC-C Delivery specification, which says that Delivery should be processed as an asynchronous request. We kept that deviation of the py-tpcc implementation, so CH2 and CH2++ are non-compliant (by design) in that they process Delivery operations synchronously. CH2++'s operational performance is reported in terms of throughput and response times for NewOrder operations.

3.4 CH2++ Analytical Queries

When we designed the analytical half of CH2's workload, we started with the 22 queries of CH, which were inspired by TPC-H's 22 queries. We modified CH's queries to operate meaningfully against the historical data discussed in Section 3.2 with selectivities like those of TPC-H: We replaced baked-in constants with randomly generated controlled values and we examined and modified query predicates to get 22 queries that were much better aligned with TPC-H's query characteristics [7]. The queries were expressed in SQL++, a generalization of SQL that handles nested and schemaless data [6,8]. For CH2++, we reformulated the queries from CH2 to operate against CH2++'s JSON schema, which involved dealing with its nested objects and nested arrays. Here we will discuss three of the 22 queries to illustrate some of the key differences between CH2++'s queries and their CH and CH2 predecessors. The complete CH2 and CH2++ query sets are available at `https://github.com/couchbaselabs/ch2`.

Figure 2 shows Query 1 of the CH2++ query set (on the right) along with the original relational CH query (on the left). For this first query, the CH2++ version is the same as CH2. The differences between the relational CH query and the CH2++ version are highlighted. The first difference is due to the JSONic nesting of orderlines in orders; the second difference is the replacement of a (non-meaningful) constant date with a parameterized date range.

```
SELECT ol_number,
  SUM(ol_quantity) AS sum_qty,
  SUM(ol_amount) AS sum_amount,
  SUM(ol_quantity) AS avg_qty,
  AVG(ol_amount) AS avg_amount,
  COUNT(*) AS count_order
FROM orderline
WHERE ol_delivery_d >
  '2007-01-02 00:00:00.000000'
GROUP BY ol_number
ORDER BY ol_number
```

```
SELECT ol.ol_number,
  SUM(ol.ol_quantity) AS sum_qty,
  SUM(ol.ol_amount) AS sum_amount,
  AVG(ol.ol_quantity) AS avg_qty,
  AVG(ol.ol_amount) AS avg_amount,
  COUNT(*) AS count_order
FROM orders o, o.o_orderline ol
WHERE ol.ol_delivery_d >
  DATE_ADD_STR('[START_DATE]',[DAYS],'day')
GROUP BY ol.ol_number
ORDER BY ol.ol_number;
```

Fig. 2. Query 1 – CH (left) vs. CH2 / CH2++ (right)

Figure 3 shows Query 3 from the new CH2++ query set (right) alongside the corresponding version from CH2 (left). In this query, one can see the impact of having the nested array of addresses for customers, with the shipping address being the one of interest: The CH2++ query unnests the array (as variable *ca*) and singles out the shipping address. This is the sort of complexity that can arise in non-1NF JSON analytics scenarios. As a slightly more complex example, Figure 4 offers a side-by-side comparison for Query 10. Since the CH2++ customers have nested arrays of phones as well as addresses, more unnesting and singling-out is needed to answer Query 10's business question using the shipping address and contact phone number. Similar changes were made for each of CH2's analytical queries to create CH2++'s analytical query set. Again, all 22 queries for CH2 and CH2++ can be found at `https://github.com/couchbaselabs/ch2`.

```
SELECT o.o_id, o.o_w_id, o.o_d_id,
  SUM(ol.ol_amount) AS revenue,
  o.o_entry_d
FROM customer c,
  neworder no,
  orders o, o.o_orderline ol
WHERE c.c_state LIKE '[CSTATE]%'

  AND c.c_id = o.o_c_id
  AND c.c_w_id = o.o_w_id
  AND c.c_d_id = o.o_d_id
  AND no.no_w_id = o.o_w_id
  AND no.no_d_id = o.o_d_id
  AND no.no_o_id = o.o_id
  AND o.o_entry_d <
    '[O_YEAR]-[O_MONTH]-[O_DAY]' ||
    '00:00:00.000000'
GROUP BY o.o_id, o.o_w_id, o.o_d_id,
  o.o_entry_d
ORDER BY revenue DESC, o.o_entry_d;
```

```
SELECT o.o_id, o.o_w_id, o.o_d_id,
  SUM(ol.ol_amount) AS revenue,
  o.o_entry_d
FROM customer c, c.c_addresses ca,
  neworder no,
  orders o, o.o_orderline ol
WHERE ca.c_state LIKE '[CSTATE]%'
  AND ca.c_address_kind = 'shipping'
  AND c.c_id = o.o_c_id
  AND c.c_w_id = o.o_w_id
  AND c.c_d_id = o.o_d_id
  AND no.no_w_id = o.o_w_id
  AND no.no_d_id = o.o_d_id
  AND no.no_o_id = o.o_id
  AND o.o_entry_d <
    '[O_YEAR]-[O_MONTH]-[O_DAY]' ||
    '00:00:00.000000'
GROUP BY o.o_id, o.o_w_id, o.o_d_id,
  o.o_entry_d
ORDER BY revenue DESC, o.o_entry_d;
```

Fig. 3. Query 3 – CH2 (left) vs. CH2++ (right)

```
 SELECT c.c_id,
   c.c_last,
   SUM(ol.ol_amount) AS revenue,
   c.c_city, c.c_phone,
   n.n_name
FROM customer c,

  orders o, o.o_orderline ol,
  nation n
WHERE c.c_id = o.o_c_id
  AND c.c_w_id = o.o_w_id
  AND c.c_d_id = o.o_d_id
  AND  o.o_entry_d >=
    '[O_YEAR]-[O_MONTH]-[O_DAY]' ||
    '00:00:00.000000'
  AND o.o_entry_d <
    DATE_ADD_STR(
      '[O_YEAR]-[O_MONTH]-[O_DAY]' ||
    '00:00:00.000000', 3, 'month'
    )


  AND n.n_nationkey =
    string_to_codepoint(c.c_state)[0]
GROUP BY c.c_id,
  c.c_last,
  c.c_city, c.c_phone,
  n.n_name
ORDER BY revenue DESC
LIMIT 20;
```

```
SELECT c.c_id,
  c.c_name.c_last,
  SUM(ol.ol_amount) AS revenue,
  ca.c_city, cp.c_phone_number,
  n.n_name
FROM customer c,
  c.c_addresses ca, c.c_phones cp,
  orders o, o.o_orderline ol,
  nation n
WHERE c.c_id = o.o_c_id
  AND c.c_w_id = o.o_w_id
  AND c.c_d_id = o.o_d_id
  AND o.o_entry_d >=
    '[O_YEAR]-[O_MONTH]-[O_DAY]' ||
    '00:00:00.000000'
  AND o.o_entry_d <
    DATE_ADD_STR(
      '[O_YEAR]-[O_MONTH]-[O_DAY]' ||
    '00:00:00.000000', 3, 'month'
    )
  AND ca.c_address_kind = 'shipping'
  AND cp.c_phone_kind = 'contact'
  AND n.n_nationkey =
    string_to_codepoint(ca.c_state)[0]
GROUP BY c.c_id,
  c.c_name.c_last,
  ca.c_city, cp.c_phone_number,
  n.n_name
ORDER BY revenue DESC
LIMIT 20;
```

Fig. 4. Query 10 – CH2 (left) vs. CH2++ (right)

The analytical performance of CH2++ is reported in terms of the geometric mean ("power") of the response times of the 22 queries when they are looped and run in a sequence. The order of execution for the 22 analytical queries follows the permutation specified at the end of the TPC-H specification [25].

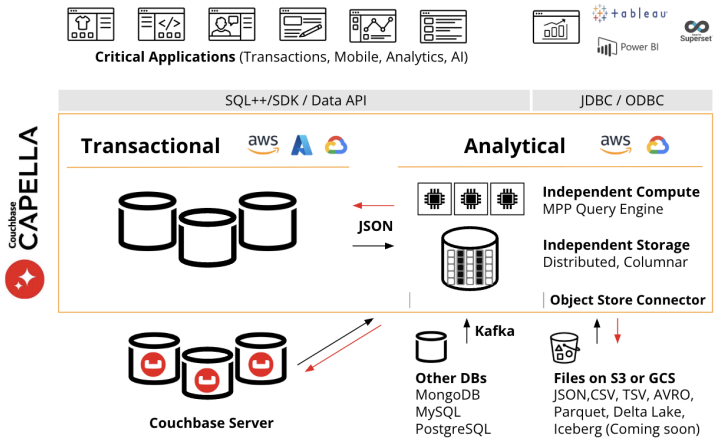


Fig. 5. Couchbase Cappella Service

4 A DBaaS Target: Couchbase Capella

To show the utility of CH2++, we apply it here to Couchbase Capella, a scalable and HOAP-ful document DBaaS offering [13]. Figure 5 is a high-level overview of Couchbase Capella. As a managed cloud version of the scalable, shared-nothing Couchbase Server platform [3], the Capella Operational service includes a fast key-value store for sub-millisecond data operations, secondary indexing for fast querying, and a SQL++-based query engine. Its target workloads are online applications that need low query latency and high throughput. The complementary new Capella Columnar service [12] brings HOAP to Capella by providing a much-extended version of Couchbase Analytics [15] to support more complex analytical SQL++ queries against JSON data drawn from a variety of sources, including Capella Operational data, in a highly parallel manner.

Figure 6 shows the main components of the Couchbase Operational service. Internally, the Operational service is organized as a set of microservices deployed and managed on a Couchbase Operational cluster. Nodes can be added or removed via a rebalance process that redistributes data across all nodes to increase or decrease the CPU, memory, and/or storage capacity of a cluster. A homogeneous cluster of 8 nodes is shown, but applications with advanced performance needs can map the microservices to different numbers of nodes; this option is referred to as Multi-Dimensional Scaling. Internally, data changes are communicated across services via an internal Database Change Protocol (DCP) that notifies other services of changes to the documents managed by the Data Service.

The Data Service provides caching, persistence, and inter-node replication. The data model is JSON, and documents are stored in containers called buckets. There are no explicitly defined schemas, so the "schema" for documents is captured in the structure of each stored document. Developers can thus add new objects and properties at any time by deploying new application code that stores new JSON data without having to make and deploy changes to a static

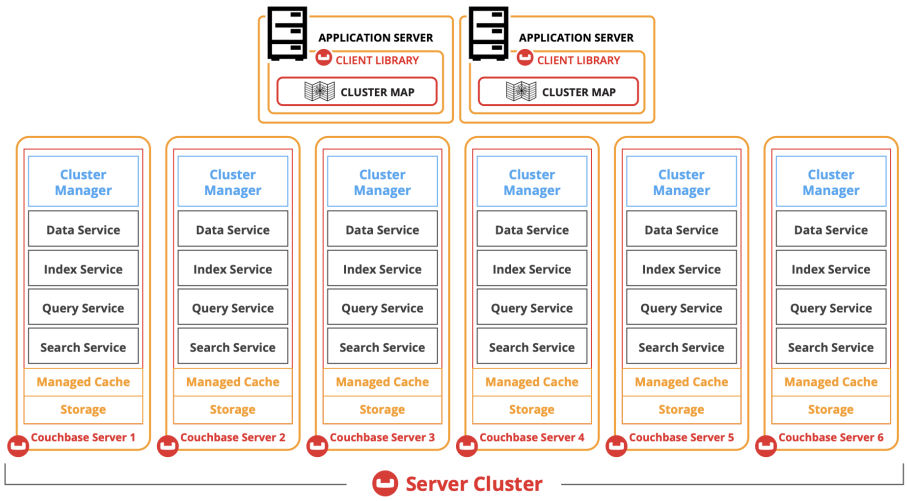


Fig. 6. Capella Operational Server Components

schema. Within a bucket, documents reside in collections (similar to RDBMS tables) that are grouped logically using scopes (similar to RDBMS schemas).

The Indexing, Query, and Search Services work together to provide JSON database functionality. The Indexing Service provides global secondary indexing for data in the Data Service, and the Search service offers richer text indexing and search. The Query Service ties things together by exposing database functionality through SQL++, a declarative, SQL-based language that relaxes the 1NF and strong typing demands of SQL. It includes support for SQL-style transactions using a combination of optimistic and pessimistic concurrency control. SQL++ statements can be grouped into atomic transactions whose effects span the Query, Indexing, and Data Services.

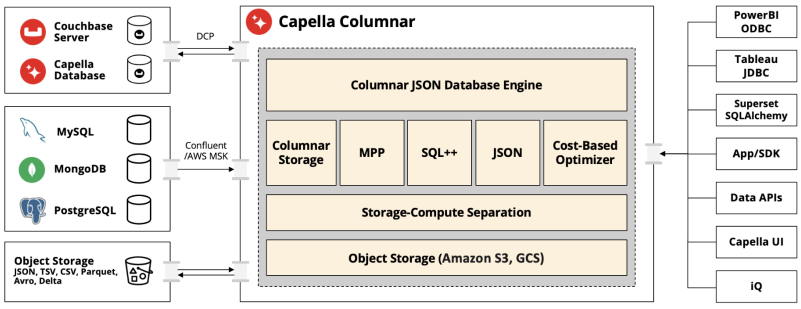


Fig. 7. Capella Columnar Service Overview

The Capella Columnar service complements the Capella Operational service by supporting more complex ad-hoc analytical queries (e.g. large joins, aggregations, windowing, etc.) over JSON document collections. Figure 7 gives a high-level overview of Capella Columnar. Whereas Capella Operational supports many users making well-defined, programmatic requests that are smaller and inexpensive, Capella Columnar focuses on ad hoc and analytical requests, with fewer users posing larger, more expensive queries. As Figure 5 indicates, Capella Columnar supports the real-time shadowing (i.e., continuous ingestion) of JSON data from Capella Operational, from on-prem Couchbase Server clusters, or from other databases, including MongoDB and relational sources. Under the hood, Capella Operational uses a largely point-to-point-based query execution model, whereas Capella Columnar utilizes partitioned-parallel processing (MPP) in order to bring all of its resources to bear on each query.

So – how does Couchbase Capella provide HOAP? Because data in Capella Operational becomes available for analysis in Capella Columnar as soon as it is created or changed, analysts always see fresh application data (thanks to DCP). They can immediately pose queries about their operational data, in its natural JSON form (with no ETL), thereby lowering the time to insight from days or hours to seconds. A Capella Columnar cluster can be scaled both horizontally and independently from its data sources. In contrast to many relational HTAP offerings, Capella Columnar is not an in-memory solution. Rather, it is designed to handle large volumes of NoSQL documents – documents whose individual worth and access frequencies would not justify the cost of a memory-resident solution, but whose aggregated content can be invaluable for decision making.

5 Benchmark Results

In this section we present results from implementing and running the proposed CH2++ benchmark on Capella's Operational and Columnar cloud services.

5.1 Benchmark Implementation

In the Capella Operational service, the CH2++ data was in a scope called *ch2pp* in a bucket called *bench*. Data Definition 1.1 shows the DDL statements for creating the benchmark's collectionsm and Data Definition 1.2 shows the DDL for creating their indexes. In Capella Columnar, a shadow collection was created for each Operational collection. Data Definition 1.3 shows the DDL statements to create the shadows and Data Definition 1.4 shows the DDL statements used to index them. Data Definition 1.5 shows the Columnar DDL statements used to create collection samples to enable cost-based query optimization (CBO).

To drive the benchmark's workload, we took py-tpcc from CMU, which has also been used by MongoDB [16], modified its data generator per earlier discussions, and added a CH2++ driver to meet our new schema and mixed workload requirements.[2] Operational and analytical users are simulated by threads run-

[2] All the software artifacts associated with this paper's benchmark can be found at `https://github.com/couchbaselabs/ch2`.

```
CREATE SCOPE bench.ch2pp;
CREATE COLLECTION bench.ch2pp.customer;
CREATE COLLECTION bench.ch2pp.district;
... (etc.) ...
```

Data Definition 1.1. Capella Operational Collection DDL

```
CREATE INDEX cu_w_id_d_id_last
 ON bench.ch2pp.customer(c_w_id, c_d_id, c_last) USING GSI;
CREATE INDEX di_id_w_id
 ON bench.ch2pp.district(d_id, d_w_id) USING GSI;
CREATE INDEX no_o_id_d_id_w_id
 ON bench.ch2pp.neworder(no_o_id, no_d_id, no_w_id) USING GSI;
CREATE INDEX or_id_d_id_w_id_c_id
 ON bench.ch2pp.orders(o_id, o_d_id, o_w_id, o_c_id) USING GSI;
CREATE INDEX or_w_id_d_id_c_id
 ON bench.ch2pp.orders(o_w_id, o_d_id, o_c_id) USING GSI;
CREATE INDEX wh_id
 ON bench.ch2pp.warehouse(w_id) USING GSI;
```

Data Definition 1.2. Capella Operational Index DDL

```
CREATE ANALYTICS COLLECTION customer ON bench.ch2pp.customer AT RemoteLink;
CREATE ANALYTICS COLLECTION district ON bench.ch2pp.district AT RemoteLink;
... (etc.) ...
```

Data Definition 1.3. Capella Columnar Collection DDL

```
CREATE INDEX customer_c_balance ON customer(c_balance:DOUBLE);
CREATE INDEX orders_entry_d ON orders(o_entry_d:STRING);
CREATE INDEX orderline_i_id
 ON orders(UNNEST o_orderline SELECT ol_i_id:BIGINT) EXCLUDE UNKNOWN KEY;
CREATE INDEX orderline_delivery_d
 ON orders(UNNEST o_orderline SELECT ol_delivery_d:STRING) EXCLUDE UNKNOWN KEY;
```

Data Definition 1.4. Capella Columnar Index DDL

```
ANALYZE COLLECTION customer WITH { "sample": "high" };
ANALYZE COLLECTION district WITH { "sample": "high" };
... (etc,) ...
```

Data Definition 1.5. Capella Columnar Collection DDL

ning on a client driver node; each constantly sends requests to the system. Up to 160 threads sent TPC-C operations to Capella Operational, with 0-1 threads sending queries to Capella Columnar. These thread counts simulate a scenario with many front-end users and a few data analysts (in this case one).

5.2 Benchmark Configuration(s)

Our objective is to use CH2++ to explore the characteristics of HOAP-ful platforms for NoSQL, including their effectiveness at providing OLTP/OLAP performance isolation for mixed workloads and their scalability when faced with a need to support more operational users or to deliver faster analytics.

We ran the CH2++ benchmark on a pair of clusters in Couchbase Capella in the AWS cloud. The Capella Operational cluster consisted of 4-8 nodes, each with 8 vCPUs, 32 GB of memory, and 1000 GB of gp3 storage. The Capella

Columnar cluster consisted of 2-8 nodes with 8 vCPUs and 32 GB of memory; each one included a 474GB NVMe local disk. We used an AWS c5.24xlarge instance with 96 vCPUs, 192 GB of memory, 100 GB gp3 storage and up to 25 Gbps of network bandwidth to run the client driver. The Capella Operational nodes were configured to have a Data+Index+Query Service combination.

Configuration	Operational Cluster Size	Columnar Cluster Size
4Q+4A	4 nodes	4 nodes
8Q+4A	8 nodes	4 nodes
4Q+8A	4 nodes	8 nodes
8Q+8A	8 nodes	8 nodes
8Q+2A	8 nodes	2 nodes

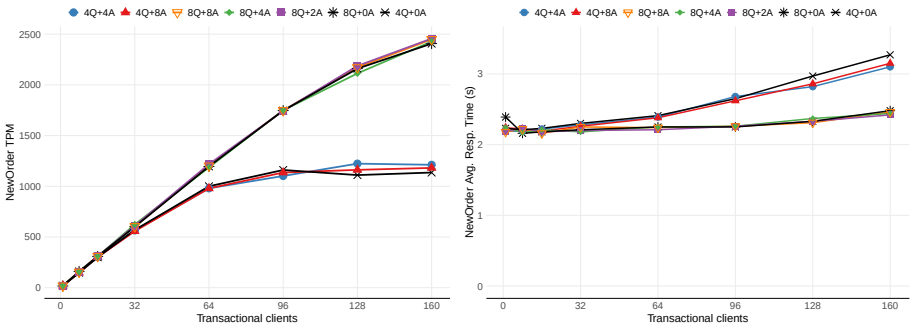
Table 2. Couchbase Capella Cluster Sizing

Similarly to what we did once for CH2 [7], we sized the two clusters in the different configurations shown in Table 2. We abbreviate the Capella Operational nodes as Query (Q) nodes since transactions are directed to its Query service. We abbreviate the Capella Columnar nodes as Analytics (A) nodes since they process (in parallel) the analytical queries. In the first configuration, 4Q+4A, 4 Operational nodes are configured with the Query, Index, and Data Services, and 4 Columnar nodes process the analytical queries. Operational requests from the client driver go to a Capella Operational API endpoint, while analytical requests go to a Capella Columnar endpoint. In the second configuration, 8Q+4A, the Operational cluster size is doubled. Symmetrically, in configuration 4Q+8A, the Columnar cluster size is instead doubled relative to its initial size. In configuration 8Q+8A, both clusters are twice their initial size. We also tried one more configuration, 8Q+2A, to test an environment with a scaled-up operational cluster feeding a scaled-down analytical cluster. The Operational and Columnar services run on disjoint cloud hardware, which in principle should ensure performance isolation for the operational workload. On each side, data is hash-partitioned across each cluster's nodes for scalability. The experiments that follow were done using a 1,000 warehouse instance of CH2++; the cardinalities of its collections are consistent with those shown earlier in Table 1.

5.3 Example Benchmark Results

We now review the findings from the 1000-warehouse mixed workload runs of CH2++. The operational and analytical clients were run concurrently until the client running the analytical queries completed one full loop through all 22 queries; the benchmark then ended and assembled its performance results.

Figure 8 shows the operational performance of CH2 ++ as a function of the number of operational client threads (4-160) for the different configurations of the cluster. In addition to the HOAPful configurations described earlier, results are shown for two configurations (8Q+0A and 4Q+0A) that have no Capella Columnar cluster (hence the 0A). Figure 8(a) shows the operational throughput



(a) Tx Clients vs NewOrder tpM (b) Tx Clients vs NewOrder RT

Fig. 8. NewOrder tpM and NewOrder RT vs. Tx Clients

in NewOrder transactions per minute. Figure 8(b) shows the corresponding average response times in seconds for the NewOrder transactions. We can make a number of important observations by examining their trends.

The first observation is that, for all configurations, Capella Operational delivers textbook performance in terms of the throughput expectations for a system with a closed workload: The throughput curves in Figure 8(a) first increase linearly and then they plateau when the system's resources become saturated, so the Operational system is *well-behaved* and it exhibits no thrashing. The response time results in 8(b) reflect this as well: The average NewOrder response times are initially flat but then they gradually become linearly proportional to the number of client threads once the system becomes saturated.

The second observation is that, if we compare the throughput curves in Figure 8(a) for a given Operational cluster size, we see *effective performance isolation*. The curves are essentially coincident despite the varying Columnar cluster sizes and presence or absence of an analytical workload. The response time results in Figure 8(b) also reflect this: The average NewOrder response time curves for a given Operational cluster size are essentially pairwise identical.

The third observation is that, if we compare the NewOrder throughput results in Figure 8(a) when the Operational cluster size is doubled (to 8Q from 4Q), the system can provide *linear transactional scaleup*. Thus, if you require twice the operational throughput, doubling your investment in "cloud hardware" can meet your requirement. In terms of the response times in Figure 8(b), we see the average response time being constant as the number of clients increases – until the cluster resources become saturated. Beyond that, response times go up linearly, and, as one would hope, a larger Operational cluster is able to keep the average response time curve flat much longer since it saturates much later.

Let us now see what CH2++ can tell us about Couchbase Capella's analytical query performance. Figure 9 shows the geometric mean of the 22 queries' average response times (a.k.a. query power) for the different cluster configurations versus the number of operational client threads. Two key take-aways are evident. First, we see the analytical side of Couchbase Capella's successful delivery of performance isolation (HOAP) – the analytical workload's query

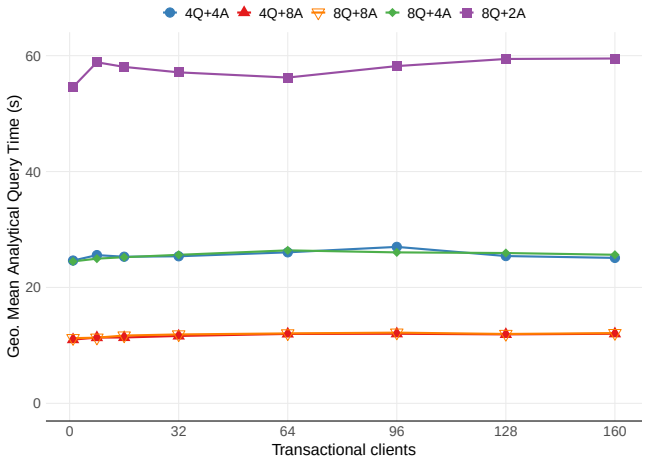


Fig. 9. Query Power (Geometric Mean of Analytical Query Times) vs. Tx Clients

performance is all but unaffected as the number of concurrent operational client threads increases. Queries are initially a little slower as the Operational cluster starts to generate updates to ingest, that is, as the number of clients first grows. This is because data ingestion uses some of the Columnar cluster's capacity, but query power is unaffected once the Operational cluster later becomes saturated. Second, we see the successful delivery of *linear query speedup*. That is, when the number of nodes given to the Columnar service is doubled, its (geometric) mean CH2++ query execution time is essentially cut in half. Thus, to get your analytical queries to run twice as fast, you can simply double your investment in the Capella Columnar cluster.

In addition to providing information on the hybrid workload handling capabilities of NoSQL platforms, CH2++ can be more broadly useful as well. As an example, Capella Columnar is a descendent of Couchbase Analytics [15]. When our earlier CH2 benchmark was created, the underlying JSON storage format was a row-oriented, self-describing binary format. Fast forward to 2025, and Capella Columnar offers both a column-oriented binary JSON format [2] as well as the previous row-oriented binary format. The newer column-oriented format is now the default format for Capella Columnar collections (hence the name). We can use CH2++ to explore the impact of the chosen storage format on JSON analytical query performance.

Figure 10 presents individual CH2++ query response times for both formats for the 4Q+4A configuration. We see significant format-related gains under the columnar format for most queries, with roughly equivalent performance in a few cases (e.g., Q11 and Q20) and a performance loss in just one case (Q22). Table 3 shows the overall 4Q+4A performance metrics for the two format options. (The ingestion rate measurement is based on the time required for Capella Columnar to ingest the initial set of 1000-warehouse-scale CH2++ collections from the corresponding Capella Operational collections.) We see that the data ingestion rate is slightly lower for the columnar format, as it has to tear apart and reformat

the incoming objects as it persists them. However, we also see a factor of two improvement in overall query performance since the columnar format allows most queries to read less data (i.e., to avoid I/O for most unreferenced columns).

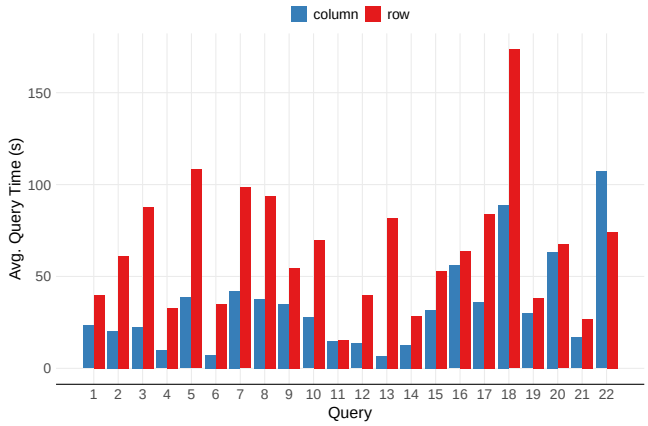


Fig. 10. Format Impact on Analytical Query Times (4Q+4A)

Storage Format	Avg. Data Ingestion Rate	Geo. Mean Query Time (Power)	Query Throughput
Column	200,633 items/sec	26.06 sec	106.85 queries/hr
Row	214,138 items/sec	55.81 sec	55.72 queries/hr

Table 3. Format Impact on Overall Columnar Performance (4Q+4A)

6 Conclusion

Systems that support hybrid workloads (HTAP or HOAP systems) first appeared in the relational world, often linked to technology trends like columnar storage and memory-rich, many-core, scale-up servers. In this paper we introduced CH2++, a major revision of our earlier CH2 benchmark, which is very well-suited to evaluate NoSQL systems that offer support for both JSON document management and JSON analytics. Like CH and CH2 before it, CH2++ borrows from and extends TPC-C and TPC-H. CH2++'s differences from CH2 include a much more document-oriented schema, the inclusion of additional fields to exercise analytical systems' column-oriented features, and a corresponding JSON "do over" of the CH2 queries. We have shared performance isolation and scaling results obtained by running a 1000-warehouse instance of CH2++ against the Capella Operational and Capella Columnar services, Couchbase's cloud service offerings. We also shared some results regarding Capella Columnar's row-

vs. column-oriented JSON storage format choice. The results have hopefully demonstrated the value of CH2++ for studying and evaluating the performance of JSON-oriented HOAP platforms. In the future, we would encourage others to try applying the CH2++ benchmark to other HOAP-ful NoSQL platforms.

References

1. 451 Research: Hybrid processing enables new use cases (business impact brief) (2018), `https://www.intersystems.com/isc-resources/wp-content/uploads/sites/24/Hybrid_Processing_Enables_New_Use_Cases-451Research.pdf` [Online; accessed 19-October-2020]
2. Alkowaileet, W., Carey, M.: Columnar formats for schemaless LSM-based document stores. PVLDB **15**(10) (2022)
3. Borkar, D., et al.: Have your data and query it too: From key-value caching to big data management. In: Proc. ACM SIGMOD Conf. (2016)
4. Carey, M., Alkowaileet, W., DiGeronimo, N., Gupta, P., Smotra, S., Westmann, T.: Towards principled, practical document database design. PVLDB (to appear)
5. Carey, M.: AsterixDB mid-flight: A case study in building systems in academia. In: 35th IEEE Int'l. Conf. on Data Eng., Macao, China, April 8-11, 2019 (2019)
6. Carey, M., Chamberlin, D., Goo, A., Ong, K.W., Papakonstantinou, Y., Suver, C., Vemulapalli, S., Westmann, T.: SQL++: we can finally relax! In: 40th IEEE Int'l. Conf. on Data Eng., Utrecht, The Netherlands, May 13-16, 2024 (2024)
7. Carey, M., Lychagin, D., Muralikrishna, M., Sarathy, V., Westmann, T.: Ch2: A hybrid operational/analytical processing benchmark for NoSQL. In: Nambiar, R., Poess, M. (eds.) Performance Evaluation and Benchmarking. pp. 62–80. Springer International Publishing (2022)
8. Chamberlin, D.: SQL++ for SQL Users: A Tutorial. Couchbase, Inc. (Available via Amazon.com.) (2018)
9. Coelho, F., Paulo, J., Vilaca, R., Pereira, J., Oliveira, R.: HTAPBench: Hybrid transactional and analytical processing benchmark. In: Proc. 8th ACM/SPEC Int'l. Conf. on Performance Eng. ICPE '17, ACM, New York, NY, USA (2017)
10. Cole, R.L., et al.: The mixed workload CH-benCHmark. In: Proc. Fourth Int'l. Workshop on Testing Database Systems, DBTest 2011, Athens, Greece, June 13, 2011. ACM (2011)
11. Cooper, B.F., et al.: Benchmarking cloud serving systems with YCSB. In: Proc. 1st ACM Symp. on Cloud Computing (SoCC), Indianapolis, Indiana, USA, June 10-11, 2010 (2010)
12. Couchbase, Inc.: About Capella Columnar (2025), `https://docs.couchbase.com/columnar/intro/intro.html`, [Online; accessed 07-June-2025]
13. Couchbase, Inc.: Welcome to Couchbase Capella (2025), `https://docs.couchbase.com/home/cloud.html`, [Online; accessed 07-June-2025]
14. Gray, J. (ed.): The Benchmark Handbook for Database and Transaction Systems (1st Edition). Morgan Kaufmann (1991)
15. Hubail, M.A., et al.: Couchbase Analytics: NoETL for scalable NoSQL data analysis. PVLDB **12**(12) (2019)
16. Kamsky, A.: Adapting TPC-C benchmark to measure performance of multi-document transactions in MongoDB. PVLDB **12**(12) (2019)
17. Kemper, A., Neumann, T.: Hyper: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In: 2011 IEEE 27th Int'l. Conf. on Data Eng. (2011)

18. Lahiri, T., et al.: Oracle database in-memory: A dual format in-memory database. In: 2015 IEEE 31st Int'l. Conf. on Data Eng. (2015)
19. Larson, P., et al.: Real-time analytical processing with SQL server. PVLDB **8**(12) (2015)
20. Li, G., Zhang, C.: HTAP databases: What is new and what is next. In: Proc. 2022 Int'l. Conf. on Management of Data. ACM, New York, NY, USA (2022)
21. May, N., Böhm, A., Lehner, W.: SAP HANA – the evolution of an in-memory DBMS from pure OLAP processing towards mixed workloads. In: Proc. BTW 2017, 17. Fachtagung des GI-Fachber. DBIS, März 2017, Stuttgart, Germany (2017)
22. Özsu, M.T., Valduriez, P.: Principles of Distributed Database Systems, 4th Edition. Springer (2020)
23. Pirzadeh, P., Carey, M., Westmann, T.: BigFUN: A performance study of big data management system functionality. In: 2015 IEEE Int'l. Conf. on Big Data (2015)
24. Pirzadeh, P., Carey, M., Westmann, T.: A performance study of big data analytics platforms. In: 2017 IEEE Int'l. Conf. on Big Data (2017)
25. Pöss, M., Floyd, C.: New TPC benchmarks for decision support and web commerce. SIGMOD Record **29**(4) (2000)
26. Pöss, M., et al.: TPC-DS, taking decision support benchmarking to the next level. In: Proc. ACM SIGMOD Conf. (2002)
27. Raab, F.: TPC-C – The standard benchmark for online transaction processing (OLTP). In: Gray, J. (ed.) The Benchmark Handbook for Database and Transaction Systems (2nd Edition). Morgan Kaufmann (1993)
28. Raman, V., et al.: DB2 with BLU acceleration: So much more than just a column store. PVLDB **6**(11) (2013)
29. Raza, A., et al.: Adaptive HTAP through elastic resource scheduling. In: Proc. ACM SIGMOD Conf. (2020)
30. Sadalage, P.J., Fowler, M.: NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley, Upper Saddle River, NJ (2013)
31. Shasha, D.E.: Database Tuning – A Principled Approach. Prentice-Hall (1992)
32. Tian, Y., Carey, M., Maxon, I.: Benchmarking HOAP for scalable document data management: A first step. In: 2020 IEEE Int'l. Conf. on Big Data (2020)
33. Wikipedia contributors: Hybrid transactional/analytical processing. Wikipedia, the free encyclopedia (2025), `https://en.wikipedia.org/w/index.php?title=Hybrid_transactional/analytical_processing&oldid=981969658`, [Online; accessed 07-June-2025]
34. Zhang, C., Li, G., Lv, T.: HyBench: A new benchmark for HTAP databases. PVLDB **17**(5) (Jan 2024)