Benchmarking HOAP for Scalable Document Data Management: A First Step

Yifan Tian
University of California, Irvine
yifant@uci.edu

Michael Carey
University of California, Irvine
mjcarey@ics.uci.edu

Ian Maxon
University of California, Irvine
imaxon@ics.uci.edu

***Abstract*—Enterprises today are becoming ever more reliant on real-time information and analytics for steering and optimizing their businesses. As a result, database system architectures with hybrid data management support – known as HTAP (Hybrid Transactional/ Analytical Processing) or HOAP (Hybrid Operational/Analytical Processing) support – are appearing and increasingly gaining traction in both the commercial and research sectors. Hybrid platforms first appeared in the relational world, and they are often linked in that world to concurrent high-end server technology trends such as columnar storage and main-memory data management.**

This paper focuses on hybrid platforms, but in a very different world – in the document data management, or NoSQL, world. In this work, we report on a first effort to characterize the hybrid performance of a scalable document database system that purports to provide what one might call ”HOAP for JSON”. We have borrowed from and extended the TPC-C benchmark to study the performance of Couchbase Server, a horizontally scalable NoSQL platform that offers HOAP via the combination of its Data/Index/Query and Analytics Services. Our results attest to the importance of architecting a NoSQL platform for HOAP, both in terms of its approach(es) to query processing and its provision of performance isolation for the operational and analytical components of a mixed workload. We share our initial results, the insights that we have gained thus far, and our thoughts on future work related to benchmarking such systems.

***Index Terms*—NoSQL, OLTP, OLAP, HTAP, HOAP, N1QL**

I. Introduction

In today’s increasingly online world, businesses and other organizations are becoming more and more reliant on real-time information and its analysis to steer and optimize their operations. Historically, operational (OLTP) and analytical (OLAP) processing were separate activities, with each running on their own separate infrastructures; periodic ETL processes served to bridge these worlds in the overall architecture of a typical enterprise [14]. Today, database system architectures with hybrid data management support – referred to as HTAP (Hybrid Transactional/Analytical Processing [23]) or HOAP (Hybrid Operational/Analytical Processing [1]) support – are appearing on the scene and gaining traction in both industry and research in order to address the pressing need for timely analytics. Originating in the relational world, hybrid platforms are commonly linked to other concurrent high-end server technology trends; columnar storage and main-memory data management are two of the technologies that are often assumed to be part of that picture.

While relational databases still dominate the enterprise IT landscape, today’s mission-critical applications demand support for millions of interactions with end-users via the Web and mobile devices. In contrast, traditional relational database systems were built to target thousands of users. Designed for strict consistency and data control, relational database systems tend to fall short of the agility, flexibility, and scalability demands of today’s new applications. This has led to the emergence of the new generation of data management systems now known as NoSQL database systems [22]; our focus here will be on the NoSQL sub-category of document databases. Examples of such systems include MongoDB [9] and Couchbase Server [3]. NoSQL systems aim to scale incrementally and horizontally on clusters of computers as well as to reduce the mismatch between the applications’ view of data and its persisted view, thereby enabling the players – ranging from application developers to DBAs and data analysts as well – to work with their data in its natural form.

Given this state of affairs, a natural question arises: Is there HOAP[1] for NoSQL, especially document databases? The answer is yes. The requirement to combine operational and analytical capabilities to support timely analytics is no less present in the NoSQL world than in the SQL world. As a result, NoSQL vendors are also beginning to focus on providing their enterprise customers with HOAP and are including HOAP-ful messages in their marketing materials. One such vendor, the one whose document data management technology we will be examining here, is Couchbase; the Couchbase Server platform introduced HOAP with its addition of Couchbase Analytics [8]. It it important to note that our benchmark design is not specific to Couchbase, however, as it could be run on HOAP-ful configurations of other NoSQL databases as well.

In the relational world, the problem of evaluating platform performance under mixed OLTP/OLAP workloads has attracted attention in recent years. One example is the mixed workload CH-benCHmark [4] proposed by a stellar collection of industrial database query processing and performance experts and now used by others to assess the performance of new HTAP systems [21]. The same is not yet true in the NoSQL

[1]For the remainder of this paper we will prefer the term HOAP in the context of NoSQL, as HOAP seems a bit less tied than HTAP to strict ACID transactions and to its tendency to attract columnar, main-memory technology presumptions.

world; there has yet to be a benchmark proposed to assess HOAP for scalable NoSQL systems. This paper reports on a first step down that road – we report a set of initial results from an attempt to evaluate the level of HOAP-fulness offered by the architecture of Couchbase Server.

The remainder of this paper is organized as follows: Section II briefly surveys related work on HTAP systems and both SQL and NoSQL benchmarks. Section III provides an overview of Couchbase Server and its approach to offering HOAP. Section IV describes our initial attempt to provide an informative HOAP benchmark. Section V presents a collection of results obtained by running our benchmark on a Couchbase server cluster under several different service configurations, Section VI summarizes our initial learnings and our thoughts on HOAP for the future.

II. Related Work

In order to set the stage for our initial work on a HOAP-ful benchmark for NoSQL, we must first review the most closely related work on HTAP/HOAP and on database benchmarks.

A. HTAP (HOAP)

As mentioned in the Introduction, the relational database world has witnessed the emergence of HTAP capabilities in a number of vendors' systems in recent years as well as growing interest in tackling research problems related to the delivery of effective HTAP solutions. Notable HTAP offerings today include such systems as HyPer [10] (born in research, but now owned by and used in Tableau) and SAP-HANA [13]. Other significant commercial relational HTAP offerings include DB2 BLU from IBM [20], Oracle's dual-engine main-memory database solution [11], and the real-time analytical processing capabilities now found in Microsoft's SQL Server [12]. On the research side, a very recent paper introduces and explores the concept of adaptive HTAP and how to manage the core and memory resources of a powerful (scale-up) many-core NUMA server running a mixed main-memory workload. [21].

Stepping back, one finds that research and development in the relational HTAP world have focused heavily on in-memory scenarios for relatively ”small” operational databases. That is, now that many-core servers with very large main memories are available, and given the degree of compression enabled by columnar storage, it is possible for main memory to hold all of an enterprises' operational business data. As a result, most current HTAP database offerings rely on main-memory database technologies. And, as would then be expected, the focus of these offerings is on single-server – i.e., scaling up rather than scaling out – architectures.

In contrast to relational HTAP, providing HOAP for scalable NoSQL document databases brings different problems that require different solutions. To scale document databases while providing HOAP, the focus needs to be more on Big Data – flexible, schema-less data. In addition, NoSQL systems and applications tend to have different transactional consistency needs [22]. Data timeliness is equally important in the NoSQL world, but there is less of a need to focus on the reduction or elimination of ACID transaction interference and more of a need to focus on the successful provision of performance isolation at the level of a cluster's physical resources (as we will see).

B. Benchmarks

There have been numerous prior benchmarks developed to evaluate the performance of relational database systems under a variety of application scenarios [7]. The most notable series of benchmarks are the TPC-x benchmarks developed over the years by the Transaction Processing Council (TPC). These include TPC-C [19] for a typical transaction processing workload as well as TPC-H [17] and TPC-DS [18] for decision support and analytics. There have also been a number of benchmarks proposed and employed in the NoSQL world, including YCSB [5] for key-value store workloads, BigFUN [15] for basic Big Data management platform operations' performance, MongoDB's recent adaptation of the TPC-C Benchmark to evaluate NoSQL transactional performance [9], and a philosophically similar NoSQL adaptation of the TPC-H benchmark [16] to evaluate Big Data analytics performance, to name a handful of the benchmarks in the NoSQL and Big Data arenas. Work of potential interest to readers on workload-independent benchmarking frameworks includes [6] and [2].

To evaluate HTAP systems, a particularly notable effort was the development of the mixed workload CH-benCHmark [4]. This benchmark resulted from a Dagstuhl workshop attended by a group of industrial database query processing and performance experts drawn from a variety of companies. The benchmark combines ideas and operations from the TPC-C and TPC-H benchmarks in order to bridge the gap between the established single-workload benchmark suites of TPC-C, for OLTP, and TPC-H, for OLAP, thus providing a foundation for mixed workload performance evaluation. The original paper included some illustrative first results from applying the benchmark to PostgreSQL with all data in memory and a read-committed isolation level. The CH-benCHmark appears to be gaining traction today for HTAP use, having very recently been used to assess the performance of a new HTAP system and its scheduling ideas [21].

To the best of our knowledge, there has yet to be a mixed NoSQL workload benchmark proposed to assess HOAP for scalable NoSQL systems.

III. Couchbase Server

Couchbase Server is a highly scalable document-oriented database management system [3]. With a shared-nothing architecture, it exposes a fast key-value store with a managed cache for sub-millisecond data operations, secondary indexing for fast querying, and a high-performance query engine (actually now two complementary engines [8]) for executing declarative SQL-like N1QL[2] queries.

Figure 1 lists the major components of Couchbase Server. (Not listed is Couchbase Mobile, which extends the Couchbase

[2]N1QL is short for Non-1NF Query Language.

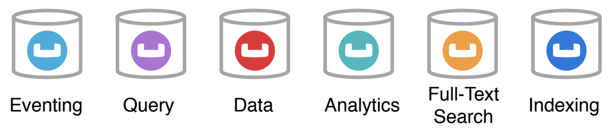


Fig. 1. Major Couchbase Server Components

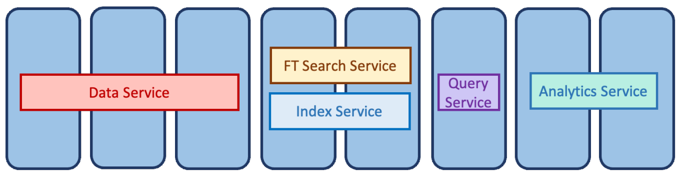


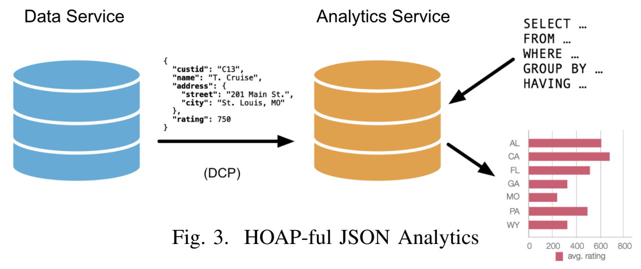
Fig. 2. Multi-Dimensional Scaling (MDS)

data platform to the edge, securely managing and syncing data from any cloud to all edge devices.) Architecturally, the system is organized as a set of services that are deployed and managed as a whole on a Couchbase Server cluster. Physically, a cluster is a group of interchangeable nodes operating in a peer-to-peer topology, and the services running on each node can be managed as required. Nodes can be added or removed through a rebalance process that redistributes the data evenly across all nodes. The addition or removal of nodes can be used to increase or decrease the CPU, memory, disk, or network capacity of a cluster. Rebalancing is done online and requires no downtime. The ability to dynamically scale the cluster and map services to sets of nodes is referred to as Multi-Dimensional Scaling (MDS). Figure 2 illustrates how MDS might enable a Couchbase cluster to be configured with three nodes for its Data Service, two nodes to be shared by its Index and Full-Text Search Services, one node for the Query Service, and two for the Analytics Service.

A key aspect of the Couchbase Server architecture, which is also relevant to HOAP, is how data mutations are communicated across services. Mutation coordination is done via an internal protocol, called the Database Change Protocol (DCP), that keeps services in sync by notifying them of all mutations to documents managed by the Couchbase Data Service. The Data Service is the producer of DCP mutation notifications; other services participate as DCP listeners.

The Data Service provides the foundation for document management. It provides caching, data persistence, and inter-node replication. The document data model is JSON, and documents are stored in containers called buckets. A bucket is a logical collection of related documents, similar to a database or a schema in a relational database. A bucket is a unique key space, and its documents can be accessed using a user-provided document ID much as one can use a primary key for lookups in an RDBMS. There is no explicitly defined schema, so the "schema" for documents in Couchbase Server is based on the application code and captured in the structure of each stored document. Developers can add new objects and properties at any time simply by deploying new application code that stores new JSON data without having to also make and deploy corresponding changes to a static schema.

The Indexing, Full-Text Search, and Query Services co-



```
{
  "custid": "C13",
  "name": "T. Cruise",
  "address": {
    "street": "201 Main St.",
    "city": "St. Louis, MO"
  },
  "rating": 750
}
```

```
SELECT …
FROM …
WHERE …
GROUP BY …
HAVING …
```

Fig. 3. HOAP-ful JSON Analytics

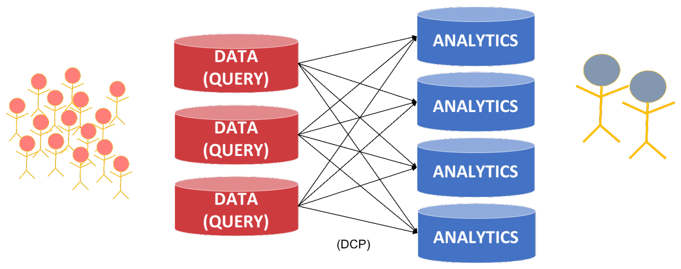


Fig. 4. Scalable Dual-Service Query Architecture

ordinate via DCP to provide user-facing document database management functionality that supports high volumes of low-latency queries and updates for JSON documents. The Indexing Service provides scalable global secondary indexing for the data managed by the Data Service, and the Full-Text Search service extends Couchbase Server's indexing capabilities to include rich text indexing and search. The Query Service ties this all together by exposing Couchbase Server's database functionality through N1QL, a declarative, SQL-based query language that relaxes the rigid 1NF and strongly-typed schema demands of the relational SQL language standard.

The Analytics Service complements the Query Service by providing support for complex and potentially expensive ad-hoc analytical queries (e.g., large joins and aggregations) over collections of JSON documents.[3] Figures 3 and 4 show the role that the Analytics Service plays in Couchbase Server. The Data and Query Service provide low-latency key-value-based and query-based access to their data. Their design point is operational; they aim to support a large number of users making well-defined, programmatic, parameterized requests that tend to be relatively small and inexpensive. In contrast, the design point for the Analytics Service is focused on ad hoc and analytical requests; it aims to support fewer users posing much larger, more expensive N1QL queries against a real-time shadow copy of the same JSON data. To this end, while the Query service has a largely point-to-point/RPC-based query execution model, the Analytics Service uses partitioned parallelism under the hood, applying best-of-breed parallel query execution techniques to bring all the resources of the Analytics nodes to bear on each query [8].

The Eventing Service provides a framework that application developers can use to respond to data changes in real time. It offers an Event-Condition-Action based model for invoking

[3]For planned analytical queries, e.g., daily reports, another option in Couchbase Server is to create a covering index for such queries. This is akin to a simple materialized view approach.

functions registered by reactive applications.

So what about HOAP? As Figures 3 and 4 aim to illustrate, operational data in Couchbase Server becomes available for analysis as soon as it is created, and analyses always see fresh application data thanks to DCP. This enables applications and data analysts to immediately pose questions against their operational data, in terms of its natural data model, reducing the time to insight from days or hours to seconds. Note that there are several differences between Couchbase Server's approach and HTAP in the relational world. One relates to scale: Like all services in Couchbase Server, the Analytics Service is designed to scale out horizontally on a shared-nothing cluster, and it can be scaled independently, as Figure 2 suggests. The Analytics Service maintains a real-time shadow copy of the operational data that the enterprise wants to have available for analysis; the copy is because Analytics is deployed on disjoint cluster nodes to provide performance isolation for the operational and analytical workloads. Another difference relates to technology: Couchbase Analytics is not an in-memory solution. It is designed to handle a large volume of NoSQL documents, documents whose individual value would not warrant an expensive memory-resident solution, but whose aggregated content is still invaluable for decision-making.

More information about Couchbase Server in general and the Analytics Service in particular can be found in references [3] and [8], respectively. The latter also provides a brief look at the relative performance of the Query and Analytics services at different data and query scales.

IV. HOAP Benchmark Design

When we undertook this study, one might say that we had ”high HOAPs”. Our goal was to explore several key aspects of NoSQL platforms' support for HOAP, including (1) their effectiveness at providing performance isolation between the OLTP and OLAP components of a mixed workload, and (2) the effectiveness of their query engines for handling OLAP-style queries. These were of particular interest because most NoSQL systems were designed to scale out horizontally on shared-nothing clusters and their initial design points for query processing have been OLTP-oriented, i.e., they were generally built to support high-concurrency/low-latency operational workloads.

Our first instinct was to try to design our own hypothetical enterprise (PetersList, modeled after Craigslist) with a mix of front-end and back-office operations. Later, with a healthy appreciation of the difficulty of coming up with our own detailed and believable schema, synthetic data, and mixed workload designs, we were attracted by what MongoDB did in extending TPC-C for evaluating transactional NoSQL system performance [9]. (In a different project we had followed a similar path by extending TPC-H for a comparative benchmark-based study of Big Data platform performance [16].) We decided to follow their path for the operational side of the workload and to design our own analytical queries, over the TPC-H schema, for the analytical side of the mix. We later happened across the mixed workload CH-benCHmark [4] for relational systems and were surprised and pleased to find that it was based on the same basic approach. A revisionist historical account of our work might describe it as a simplified first version of a NoSQL extension of the CH-benCHmark.

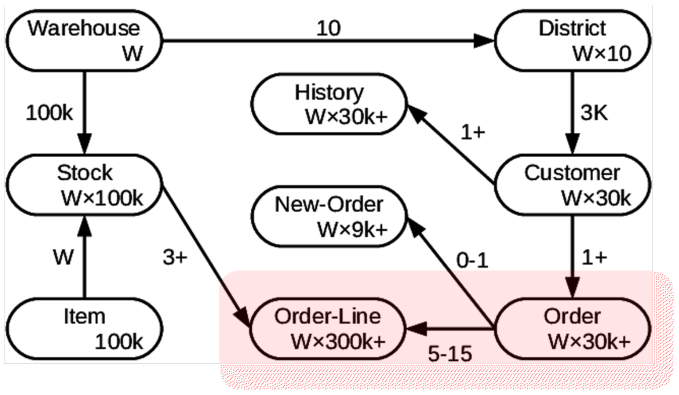


Fig. 5. TPC-C Schema (NoSQL Modification Highlighted)

A. Benchmark Database Design

The schema for our NoSQL HOAP benchmarking effort is MongoDB's adaptation [9] of the standard TPC-C schema. Figure 5 summarizes the 9 tables and relationships of the standard relational TPC-C schema. The schema models businesses which “must manage, sell, or distribute products or services” [19] and it follows a continuous scaling model. The data size is scaled based on the number of warehouses, which is increased based on the number of nodes used in the system. The MongoDB NoSQL adaptation involves 8 collections instead of 9, as in a non-1NF-limited NoSQL world, an order would naturally embed its line items as nested data. Figure 5 also highlights the affected region of the TPC-C schema. The operational workload of TPC-C models the transactions of a typical production order processing system that works against this schema. Its transactions are a mixture of read-only and update-intensive business transactions, including New-Order, Payment, Order-Status, Delivery, and Stock-Level. No other nesting changes were made to the schema, as doing so would lead to over-nesting and produce a poor database design for such use cases [9], [16].

To map this design to Couchbase Server and its services, the benchmark data was stored in its entirety in a bucket called TPCC in the Data Service. Following recommended Couchbase practices, each of the TPC-C JSON objects had an additional field, *Category*, indicating the object's type (one of ITEM, STOCK, WAREHOUSE, DISTRICT, CUSTOMER, HISTORY, NEW_ORDER, or ORDER). This bucket was the target for the operational workload's queries and updates and indexed to efficiently support them. A basic set of indexes was created by following recommendations made by the Query Service's index advisor feature. Query 1 shows the N1QL DDL statements used to set up these indexes; each index was partitioned over all of the Indexing service's cluster nodes by hashing on their indexed objects' IDs. The Analytics Service has a more fine-grained container model based on a notion of datasets (similar in granularity to MongoDB's collections).

```
CREATE INDEX idx_ITEM ON `TPCC`(I_ID)
PARTITION BY HASH(META().id) WHERE `Category` = "ITEM";

CREATE INDEX idx_WAREHOUSE ON `TPCC`(W_ID)
PARTITION BY HASH(META().id) WHERE `Category` = "WAREHOUSE";

CREATE INDEX idx_DISTRICT ON `TPCC`(D_W_ID, D_ID)
PARTITION BY HASH(META().id) WHERE `Category` = "DISTRICT";

CREATE INDEX idx_CUSTOMER ON `TPCC`(C_W_ID, C_D_ID, C_ID)
PARTITION BY HASH(META().id) WHERE `Category` = "CUSTOMER";

CREATE INDEX idx_STOCK ON `TPCC`(S_W_ID, S_I_ID)
PARTITION BY HASH(META().id) WHERE `Category` = "STOCK";

CREATE INDEX idx_NEW_ORDER
       ON `TPCC`(NO_W_ID, NO_D_ID, NO_O_ID)
PARTITION BY HASH(META().id) WHERE `Category` = "NEW_ORDER";

CREATE INDEX idx_ORDER
       ON `TPCC`(O_W_ID, O_C_ID, O_D_ID)
PARTITION BY HASH(META().id) WHERE `Category` = "ORDERS";

CREATE INDEX idx_Primary_ORDER
       ON `TPCC`(O_ID, O_W_ID, O_D_ID)
PARTITION BY HASH(META().id) WHERE `Category` = "ORDERS";

CREATE INDEX C_STATE_idx ON `TPCC`(`C_STATE`)
PARTITION BY HASH(META().id) WHERE (`Category` = "CUSTOMER");
```

Query 1. Operational Index DDL

```
CREATE DATASET tpcc_customer
       ON TPCC WHERE `Category` = "CUSTOMER";
CREATE DATASET tpcc_order
       ON TPCC WHERE `Category` = "ORDERS";
CONNECT Link Local;
```

Query 2. Analytics Dataset DDL

```
CREATE INDEX idx_customer_state
       ON tpcc_customer(C_STATE:STRING);
CREATE INDEX idx_c_idx
       ON tpcc_customer(C_ID:bigint, C_D_ID:bigint,
          C_W_ID:bigint);
CREATE INDEX idx_o_idx
       ON tpcc_order(O_C_ID:bigint, O_D_ID:bigint,
          O_W_ID:bigint);
```

Query 3. Analytics Index DDL

```
"District_find_one": 'SELECT * FROM TPCC p
WHERE p.Category = "DISTRICT"
AND D_ID = {D_ID} AND D_W_ID = {D_W_ID};',

"District_find_one_and_update": 'UPDATE TPCC p SET
p.D_NEXT_O_ID = {D_NEXT_O_ID} WHERE p.Category
= "DISTRICT" AND D_ID = {D_ID} AND D_W_ID = {D_W_ID};',

"Item_find": 'SELECT * FROM TPCC p WHERE p.Category
= "ITEM" AND I_ID IN {I_ID} AND I_W_ID = {I_W_ID};',

"Warehouse_find_one": 'SELECT * FROM TPCC p WHERE
p.Category = "WAREHOUSE" AND W_ID = {W_ID}',

"Customer_find_one": 'SELECT * FROM TPCC p WHERE
p.Category = "CUSTOMER" AND C_W_ID = {C_W_ID}
AND C_D_ID = {C_D_ID} AND C_ID = {C_ID}',

"createNewOrder": 'UPSERT INTO `TPCC` (KEY,VALUE)
VALUES("{key}", {content}) RETURNING *;',

"Stock_find": 'SELECT * FROM TPCC p WHERE p.Category
= "STOCK" AND p.S_I_ID in {S_I_ID} AND p.S_W_ID = {S_W_ID};',

"Stock_find_one": 'SELECT * FROM TPCC p WHERE p.Category
= "STOCK" AND p.S_I_ID = {S_I_ID} AND p.S_W_ID = {S_W_ID};',

"UpdateStock": 'UPDATE TPCC p SET S_QUANTITY = {S_QUANTITY},
S_YTD = {S_YTD}, S_ORDER_CNT = {S_ORDER_CNT},
S_REMOTE_CNT = {S_REMOTE_CNT} WHERE S_I_ID = {S_I_ID} AND
S_W_ID = {S_W_ID} AND Category = "STOCK";', # s_quantity,
s_order_cnt, s_remote_cnt, ol_i_id, ol_supply_w_id

"Orders_insert_one": 'UPSERT INTO `TPCC` (KEY,VALUE)
VALUES("{key}", {content}) RETURNING *;',
```

Query 4. New-Order Operation

```
SELECT c.C_ID, COUNT(*) AS OC_COUNT
FROM TPCC o JOIN TPCC c USE HASH (BUILD)
   ON o.O_C_ID = c.C_ID
   AND o.O_D_ID = c.C_D_ID
   AND o.O_W_ID = c.C_W_ID
WHERE c.Category = 'CUSTOMER'
   AND o.Category = 'ORDERS'
   AND c.C_STATE = 'ca'
GROUP BY c.C_ID
ORDER BY OC_COUNT DESC, c.C_ID ASC
LIMIT 10;
```

Query 5. Analytical query (in N1QL for Query)

```
SELECT c.C_ID, COUNT(*) AS OC_COUNT
FROM tpcc_customer c, tpcc_order o
WHERE c.C_ID = o.O_C_ID
   AND c.C_D_ID = o.O_D_ID
   AND c.C_W_ID = o.O_W_ID
   AND c.C_STATE = "ca"
GROUP BY c.C_ID
ORDER BY OC_COUNT DESC, c.C_ID ASC
LIMIT 10;
```

Query 6. Analytical query (in N1QL for Analytics)

Query 2 shows the N1QL DDL statements used to create Analytics datasets to shadow the operational data for analytical workload queries (to be described shortly). Query 3 shows the statements used to create basic indexes on these datasets in support of the analytical workload queries. Since hybrid performance trends were the focus of this first step, not absolute performance, little effort was made to tune either these initial indexing choices or the queries.

B. Benchmark Operations

On the operational side, we chose to run a concurrent mix of just one of the five transactions from TPC-C, namely New-Order. This transaction represents a complete business transaction that enters a new order with multiple nested order-lines into the database. For each order-line, 99% of the time the supplying warehouse is the home warehouse. The home warehouse is a fixed warehouse ID associated with a terminal. To simulate user data entry errors, 1% of the transactions fail and trigger a roll-back. As shown in Query 4, a New-Order transaction consists of the following sequence of steps: 1. Find one district. 2. Update the district from step 1. 3. Find one item. 4. Find one warehouse. 5. Find one customer. 6. Create a new order. 7. Find the corresponding stocks. 8. Update the corresponding stocks. 9. Insert the new order document into orders. The New-Order transaction thus touches most of TPC-C's tables and consists of both read-only queries and updates.

For the analytical side of the workload, we designed our own analytical query, one loosely inspired by the sorts of operations (joins, grouping, aggregation, and top-K ordering) that one finds in typical analytical usage. Our query was inspired by the queries in TPC-H, but cast against TPC-C's schema; it finds the top 10 customers in a given state (California, for example) based on their order volume. Queries 5 and 6 show two versions of this query. Query 5 is expressed

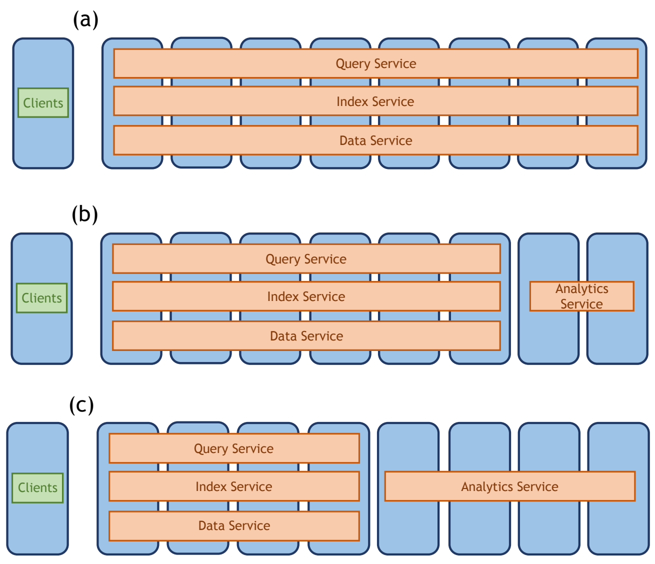


Fig. 6. Couchbase cluster configurations: (a) 0% Analytics, (b) 25% Analytics, (c) 50% Analytics.

in the N1QL for Query dialect of N1QL, and its FROM clause runs against the Data Service's TPCC bucket. Query 6 is expressed in N1QL for Analytics, and its FROM clause runs against the Analytics Service's datasets. These datasets, which were defined in Query 2, shadow the information in the TPCC bucket in real time. (Notice how the WHERE filters in their dataset definitions correspond to the WHERE predicates used to distinguish customers and orders in Query 5's version of the query.) For the analytical workload, we chose to run a concurrent mix of this query, but with the actual queried states being randomly chosen (i.e., not just California).

The overall mixed HOAP workload used in this study is a concurrent mix of the aforementioned operational and analytical queries. Note that the CH-benCHmark is a mix of *all* of TPC-C's operations on the operational side plus *all* of TPC-H's TPC-C-adapted queries on the analytica side, while the mixed workload for our first HOAP-ful step selects just one kind of operation for each side. We have done this intentionally so that we can really observe and understand the system's performance characteristics in detail in this first step.

C. Benchmark Configuration(s)

As described earlier, our goal here is to explore several key aspects of HOAP-ful platforms for NoSQL, including (1) their effectiveness at providing performance isolation between the OLTP and OLAP components of a mixed workload, and (2) the effectiveness of their query engines for handling OLAP-style queries. To this end, we have run our initial benchmark on a 9-node cluster in three different configurations. Figure 6 shows the configurations, each of which involves using one node to drive the multithreaded workload and the other nodes as an 8-node Couchbase Server cluster. In configuration (a), referred to as the 0% Analytics configuration, all of the nodes are configured to have the Data Service, Index Service, and Query Service. The operational data in the Data Service is partitioned across the whole cluster, and likewise for the Index Service's indexes; each node is able to accept and run N1QL for Query queries and updates. The Analytics Service is not deployed on the cluster in this configuration. (One might characterize this as being a HOAP-less setup.) In configuration (b), referred to as the 25% Analytics configuration, 25% of the nodes are given over to the Analytics Service, and 75% of the cluster has the Data/Index/Query Service combination deployed on it. In configuration (c), the 50% Analytics configuration, half of the cluster is given to the Analytics Service and the other half has the Data/Index/Query combination. Note that when it is deployed on a Couchbase Cluster, the Analytics Service should be given a set of nodes to itself for performance isolation.

To implement the benchmark's mixed workload we started with the publicly available MongoDB adaptation of CMU's pytpcc benchmarking system [9] and added a driver for Couchbase Server to meet our requirements[4]. Each operational or analytical user is simulated by a thread running on the client node of the 9-node cluster. Each thread consistently sends query requests to the system. 0-64 threads send New-Order operations to the Query Service, while 0-6 threads send analytical queries to either the Query Service (in the 0% Analytics configuration) or the Analytics Service (in the other two configurations). These thread counts simulate a typical business model with more front-end users than data analysts.

Hardware-wise, the 9-node cluster used for our experiments is comprised of Intel NUC nodes, each with a dual-core Intel i7-7567U 3.5GHZ CPU, 16 GB of memory, and a 500GB Samsung 960 EVO SSD drive. Note that this is nothing like a production cluster; rather, it is a small academic ”toy” cluster that cost about $500/node to assemble and lives in an office at UC Irvine.

Data-wise, the operational data resides in the TPCC bucket in the Data Service (in JSON document form). In terms of scale, the size of this bucket was chosen to be approximately 4 times the aggregate memory size of the system. To this end, the TPCC bucket consists of 159,308,800 documents in 6 ”tables” (Categories). There are 1600 Warehouse documents, 16000 District documents, 24,000,000 Customer documents, 24,000,000 Order documents (with embedded Order-Item arrays), 7,200,000 New-Order documents, 100,000 Item documents, and 80,000,000 Stock documents.

Before proceeding to share our initial experimental results, a few more words are in order regarding the details of the analytical query that we used. Initially our intention was for the analytical query in the mix to find the top-10 customers overall. However, one of our objectives was to compare the Query and Analytics N1QL query engines, which is why we have two versions of the query. Moreover, to make our experiments feasible, we set a goal of limiting the time required for the Query Service to complete the analytical query to be 30 minutes or less. To achieve this goal we ended up having to add a state condition to the query and to carefully index the query for the Query Service. (One index was built

[4]The software artifacts associated with this paper's benchmark are available at `https://github.com/YifanTian/HOAP_Benchmark`

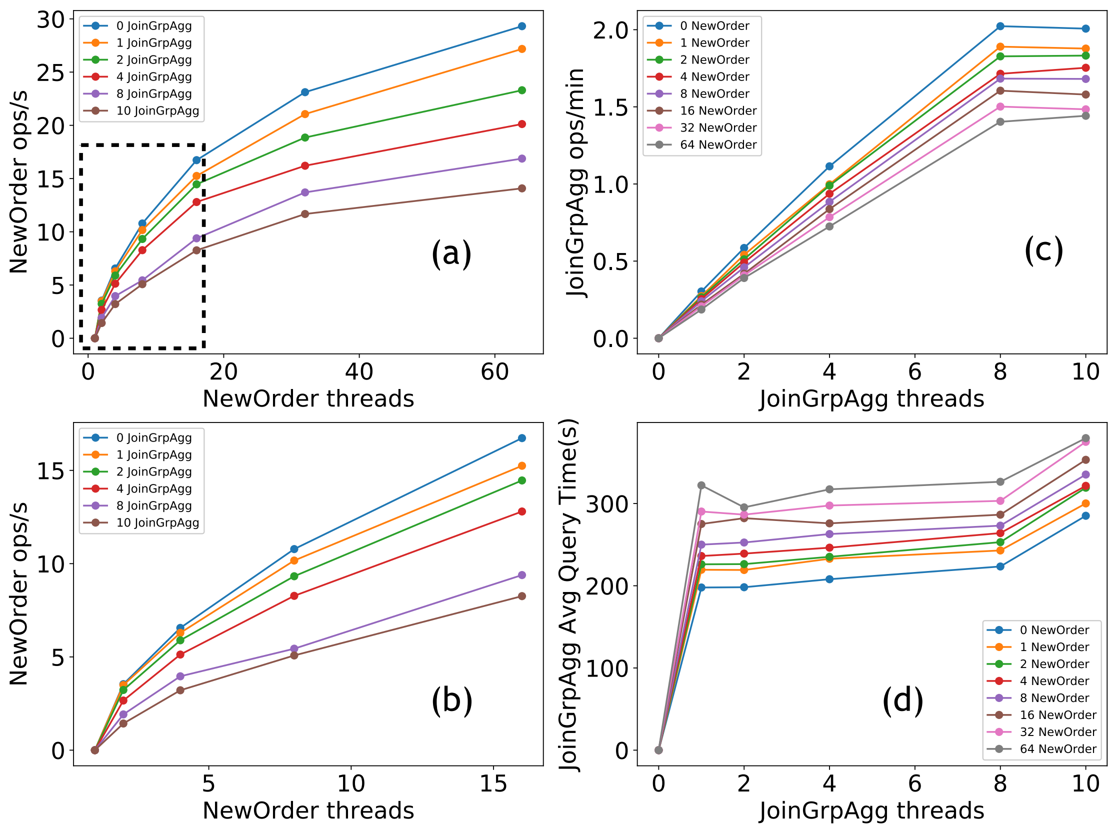


Fig. 7. 0% Analytics: (a) Operational Throughput. (b) Operational Throughput (zoomed-in). (c) Analytical Throughput. (d) Analytical Response Time.

on the customer table with the state attribute being covered to speed access to a particular state's data. A second index was built on the foreign key(s) of the Orders table to support the customer/order join operation.) Without the added predicate and the supporting indexes we kept experiencing a mix of timeouts and "too many results for index cursor" errors that impeded our ability to conduct the benchmark on the 0% Analytics configuration. (We will have a bit more to say about this when we discuss the performance results.)

V. Initial Performance Results

In this section, we report and reflect on the performance results obtained by running our initial mixed-workload HOAP benchmark on the three 8-node Couchbase Server cluster configurations shown in Figure 6. Note that our primary focus here will be on HOAP performance trends and behavior, not on the system's absolute performance. This is not production hardware, and additional benchmark tuning would no doubt be possible. For each configuration, we present a block of four performance graphs that shed light on the system's behavior with respect to operational performance, analytical performance, and performance isolation. To this end, for each configuration, we will present a block of four graphs showing the following metrics:

a. *Operational Throughput*: The throughput (in operations/second) for the New-Order operations (*NewOrder*) as a function of the number of New-Order users (threads). This graph will contain multiple curves, one for each of a different number of concurrent analytical (*JoinGrpAgg*) queries in the mix.
b. *Operational Throughput (zoomed in)*: A zoomed-in version of the dashed region of (a) to better show the initial portion of the operational throughput curves.
c. *Analytical Throughput*: The throughput (in queries/minute) for the analytical queries as a function of the number of analytical query users (threads). This graph will contain multiple curves, one for each of a different number of concurrent New-Order operations in the mix.
d. *Analytical Response Time*: The corresponding average response time (in seconds) for the analytical queries as a function of the number of analytical users, again with different numbers of concurrent New-Order operations in the mix.

The measurement period for each single mixed workload experiment (i.e., each graph data point) was 30 minutes. The benchmark data was initially loaded and it was reset to its original state before the start of each subsequent experiment. In addition, we allowed for a 5-minute cool-down interval between each experiment to make sure that each experiment began with the system at rest.

Figure 7 shows the results for the 0% Analytics configuration. In this first configuration, both the operational and analytical components of the mixed workload are directed to the Query Service and processed by the Data/Index/Query Service

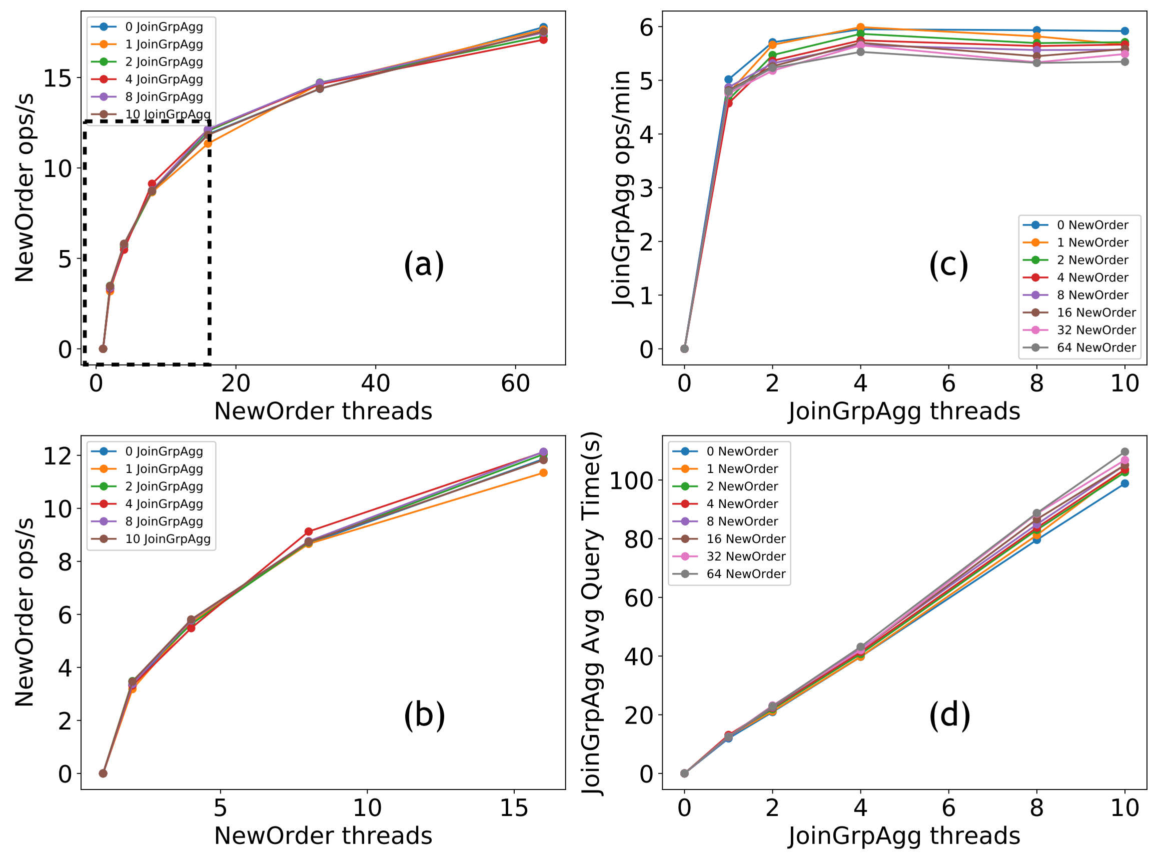


Fig. 8. 25% Analytics: (a) Operational Throughput. (b) Operational Throughput (zoomed-in). (c) Analytical Throughput. (d) Analytical Response Time.

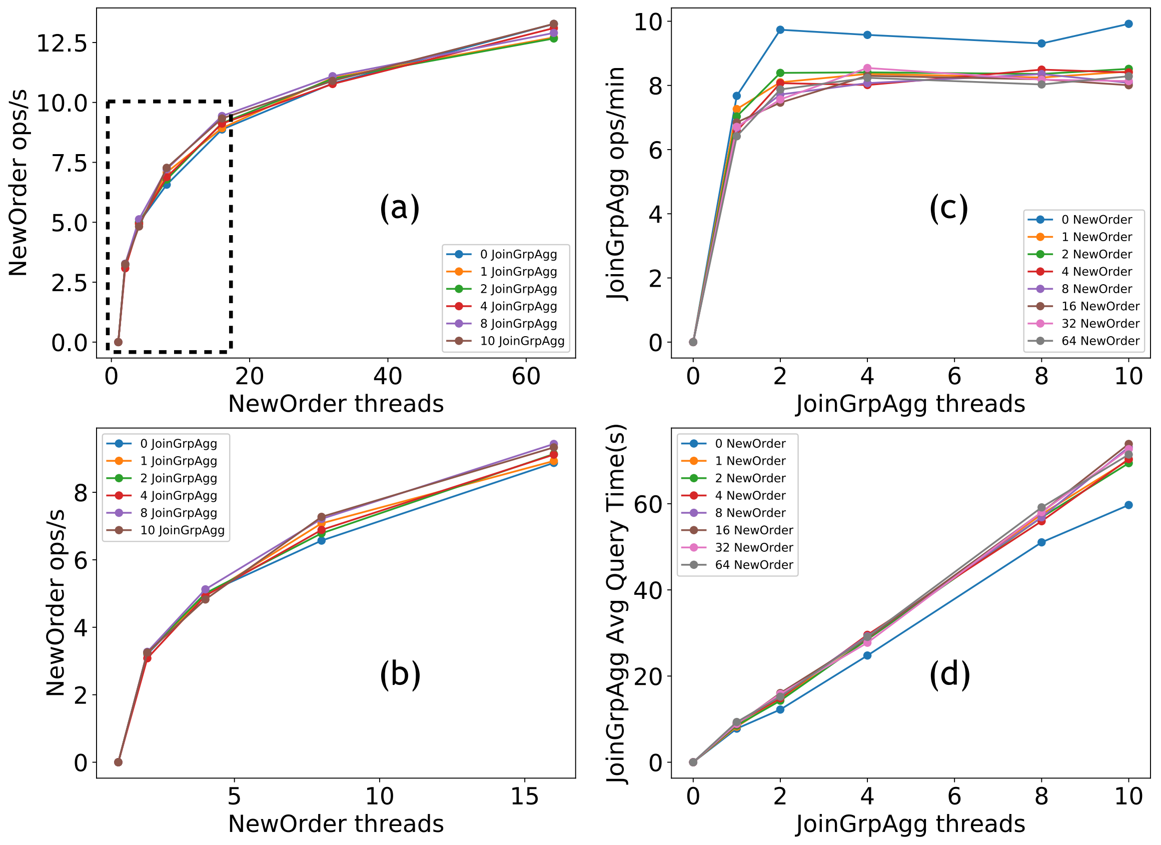


Fig. 9. 50% Analytics: (a) Operational Throughput. (b) Operational Throughput (zoomed-in). (c) Analytical Throughput. (d) Analytical Response Time.

combination. The original design point for this combination of Couchbase services is operational; this is the same service layout used for the operational performance examples in [3]. As a result, it is quite literally a ”big ask” to expect the Query Service to handle the analytical queries in the mix. (Recall from earlier that we were unable to get the Query Service to run a non-state-predicated version of this query within the designated measurement period without incurring timeouts or exceeding limits on intermediate result cursor set sizes.)

Figures 7(a) and 7(b) show the NewOrder throughput in the 0% Analytics configuration. Each of the curves exhibits a textbook performance increase as the level of NewOrder concurrency is increased. However, we can see that the operational throughput is significantly impacted by the presence of the concurrent JoinGrpAgg queries; e.g., NewOrder throughput drops by more than a factor of two when 64 concurrent NewOrder threads are competing for resources with 10 JoinGrpAgg query threads. We can also see in Figures 7(c) and 7(d) that this configuration's analytical performance is also impacted by having to compete with the NewOrder operations; e.g., the average analytical response time increases by over 50% over the range of concurrent NewOrder operations tests. These results clearly show that there is no performance isolation for the workload mix in this configuration – or to put it succinctly, they show that this configuration is essentially HOAP-less, performance-wise.

Figure 8 shows the performance results for the first HOAP-ful cluster configuration, the 25% Analytics configuration. In this configuration, 6 of the 8 nodes are running the Data/Index/Query service combination and the other 2 nodes are dedicated to running the Analytics service. The N1QL statements for the operational workload are directed to the Query Service, while the analytical workload's N1QL statements are directed to the Analytics Service. (Each has its own HTTP endpoint to which the relevant requests are sent by the threads in the HOAP benchmark's client driver.)

The differences between the results in Figures 7 and 8 are striking. Both the operational and analytics throughput curves in Figure 8 exhibit textbook increases as the number of operational or analytical threads is increased. However, unlike the 0% Analytics configuration's results, these results show essentially perfect performance isolation. By carving out 75% of the cluster for the operational services and giving the remainder to the Analytics service, the NewOrder throughput is completely protected from the JoinGrpAgg queries in the workload – social distancing works! Looking at the analytical performance results in Figure 8, we see that query performance is impacted only slightly by increases in the size of the operational portion of the workload; this makes sense because the NewOrder threads generate data changes in the Data Service's TPCC bucket that are communicated to the Analytics Service (via DCP) and applied in real-time to the Analytics Service's datasets, leading to some additional work for Analytics since it has to apply the changes.

Aside from showing the effectiveness of performance isolation, a comparison of the performance results from the two configurations also reveals some interesting information about the achievable absolute performance for the mixed workload's components. A comparison of the two configurations' NewOrder throughputs in the absence of JoinGrpAgg queries shows the ”opportunity cost” of taking 2 nodes away from the Data/Index/Query side of the cluster and giving them (unused) to the Analytics service. However, once there are 3-4 JoinGrpAgg queries present in the mix, the two configurations' operational throughputs become similar, and beyond that degree of analytical competition, the operational throughput in the 0% Analytics configuration suffers in comparison. A comparison of the two configurations' analytical query performance shows much superior performance for the 25% configuration (due to its having a parallel query engine designed for analytical query processing) – for this relatively small/simple state-predicated JoinAggGrp query, the 25% configuration delivers roughly 3X better analytical performance. (And recall that without the state predicate, the Query Service was unable to run the analytical queries within the desired experiment time limit, which was not a problem at all for the Analytics Service.)

Figure 9 shows the performance results for the final HOAP-ful cluster configuration that we tested, the 50% Analytics configuration. In this configuration, half of the nodes are running the Data/Index/Query service combination and the other half are dedicated to the Analytics service. The observations about this configuration's performance are similar to what we saw for 25% Analytics, except that in this case – due to the 50-50 split of the cluster's resources – the peak operational performance is lower and the peak analytical performance is higher due to the shift of resources in favor of the analytical side of the workload. We can also observe more clearly here the slight impact of the operational side on the analytical side; in the absence of any NewOrder threads, the analytical performance is higher than when the Analytics Service must ingest incoming mutations while also executing its JoinGrpAgg workload's queries.

VI. Conclusion

Database management systems with hybrid data management support – HTAP or HOAP – are increasingly attracting interest in both the commercial IT and research arenas. They first appeared in the relational world, where they are often linked to high-end server technology trends such as columnar storage and memory-rich, many-core scale-up server technology. In this paper we have focused on hybrid platforms in the document data management region of the NoSQL world. We reported results from a first effort to characterize the mixed workload performance of such systems. We borrowed from and extended the TPC-C benchmark and studied the performance of Couchbase Server, a horizontally scalable NoSQL platform that offers HOAP via the combination of its Data/Index/Query and Analytics Services. As we have just seen experimentally, our results attest to the importance of architecting a NoSQL platform for HOAP, both in terms of its provision of performance isolation for the operational and analytical sides of a mixed workload and its approach(es) to query processing for simple versus complex queries. We shared our initial results, which indicate that Couchbase Server is indeed able to offer "HOAP for NoSQL". We also saw that it supports configuration tuning that can enable a system administrator to choose how to split a cluster's resources between the two sides of a given mixed workload.

While HOAP-fully interesting, this paper is just a first step; many interesting things remain to be done. As discussed, our work is similar to the relational CH-benCHmark, which proposed the use of a combination of TPC-C plus a TPC-C-schema-adapted version of the TPC-H query set to form a more general mixed workload. One obvious next step would be to run the entire CH-benCHmark on Couchbase Server using the NoSQL-modified TPC-C schema and to figure out how to interpret the resulting (more complex) performance metrics. Another potentially interesting exercise would be to do this for multiple NoSQL systems in order to evaluate their relative levels of HOAP-ful-ness.

Acknowledgment

The work of the second and third authors was supported in part by NSF CNS award 1925610. Those authors also have part-time consulting affiliations with Couchbase, Inc. The authors would like to thank the BPOD 2020 reviewers and Peter Reale of Couchbase for suggestions that helped to improve the presentation. They would also like to thank Keshav Murthy of Couchbase for his occasional assistance with the Query Service.

References

[1] 451 Research. *Hybrid Processing Enables New Use Cases (Business Impact Brief)*, 2018 (accessed October 2020). https://www.intersystems.com/isc-resources/wp-content/uploads/sites/24/Hybrid_Processing_Enables_New_Use_Cases-451Research.pdf.

[2] D. Bermbach, J. Kuhlenkamp, A. Dey, A. Ramachandran, A. D. Fekete, and S. Tai. BenchFoundry: A benchmarking framework for cloud storage services. In E. M. Maximilien, A. Vallecillo, J. Wang, and M. Oriol, editors, *Service-Oriented Computing - 15th International Conference, ICSOC 2017, Malaga, Spain, November 13-16, 2017, Proceedings*, volume 10601 of *Lecture Notes in Computer Science*, pages 314–330. Springer, 2017.

[3] D. Borkar, R. Mayuram, G. Sangudi, and M. J. Carey. Have your data and query it too: From key-value caching to Big Data management. In F. Özcan, G. Koutrika, and S. Madden, editors, *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 239–251. ACM, 2016.

[4] R. L. Cole, F. Funke, L. Giakoumakis, W. Guy, A. Kemper, S. Krompass, H. A. Kuno, R. O. Nambiar, T. Neumann, M. Poess, K. Sattler, M. Seibold, E. Simon, and F. Waas. The mixed workload CH-benCHmark. In G. Graefe and K. Salem, editors, *Proceedings of the Fourth International Workshop on Testing Database Systems, DBTest 2011, Athens, Greece, June 13, 2011*, page 8. ACM, 2011.

[5] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with YCSB. In J. M. Hellerstein, S. Chaudhuri, and M. Rosenblum, editors, *Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC 2010, Indianapolis, Indiana, USA, June 10-11, 2010*, pages 143–154. ACM, 2010.

[6] D. E. Difallah, A. Pavlo, C. Curino, and P. Cudré-Mauroux. OLTP-Bench: An extensible testbed for benchmarking relational databases. *Proc. VLDB Endow.*, 7(4):277–288, 2013.

[7] J. Gray, editor. *The Benchmark Handbook for Database and Transaction Systems (1st Edition)*. Morgan Kaufmann, 1991.

[8] M. A. Hubail, A. Alsuliman, M. Blow, M. J. Carey, D. Lychagin, I. Maxon, and T. Westmann. Couchbase Analytics: NoETL for scalable NoSQL data analysis. *Proc. VLDB Endow.*, 12(12):2275–2286, 2019.

[9] A. Kamsky. Adapting TPC-C benchmark to measure performance of multi-document transactions in MongoDB. *Proc. VLDB Endow.*, 12(12):2254–2262, 2019.

[10] A. Kemper and T. Neumann. HyPer: A hybrid OLTP & OLAP main memory database system based on virtual memory snapshots. In S. Abiteboul, K. Böhm, C. Koch, and K. Tan, editors, *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, pages 195–206. IEEE Computer Society, 2011.

[11] T. Lahiri, S. Chavan, M. Colgan, D. Das, A. Ganesh, M. Gleeson, S. Hase, A. Holloway, J. Kamp, T. Lee, J. Loaiza, N. MacNaughton, V. Marwah, N. Mukherjee, A. Mullick, S. Muthulingam, V. Raja, M. Roth, E. Soylemez, and M. Zaït. Oracle database in-memory: A dual format in-memory database. In J. Gehrke, W. Lehner, K. Shim, S. K. Cha, and G. M. Lohman, editors, *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, pages 1253–1258. IEEE Computer Society, 2015.

[12] P. Larson, A. Birka, E. N. Hanson, W. Huang, M. Nowakiewicz, and V. Papadimos. Real-time analytical processing with SQL server. *Proc. VLDB Endow.*, 8(12):1740–1751, 2015.

[13] N. May, A. Böhm, and W. Lehner. SAP HANA - The evolution of an in-memory DBMS from pure OLAP processing towards mixed workloads. In B. Mitschang, D. Nicklas, F. Leymann, H. Schöning, M. Herschel, J. Teubner, T. Härder, O. Kopp, and M. Wieland, editors, *Datenbanksysteme für Business, Technologie und Web (BTW 2017), 17. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 6.-10. März 2017, Stuttgart, Germany, Proceedings*, volume P-265 of *LNI*, pages 545–563. GI, 2017.

[14] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems, 4th Edition*. Springer, 2020.

[15] P. Pirzadeh, M. J. Carey, and T. Westmann. BigFUN: A performance study of Big Data management system functionality. In *2015 IEEE International Conference on Big Data, Big Data 2015, Santa Clara, CA, USA, October 29 - November 1, 2015*, pages 507–514. IEEE Computer Society, 2015.

[16] P. Pirzadeh, M. J. Carey, and T. Westmann. A performance study of Big Data analytics platforms. In J. Nie, Z. Obradovic, T. Suzumura, R. Ghosh, R. Nambiar, C. Wang, H. Zang, R. Baeza-Yates, X. Hu, J. Kepner, A. Cuzzocrea, J. Tang, and M. Toyoda, editors, *2017 IEEE International Conference on Big Data, BigData 2017, Boston, MA, USA, December 11-14, 2017*, pages 2911–2920. IEEE Computer Society, 2017.

[17] M. Pöss and C. Floyd. New TPC benchmarks for decision support and web commerce. *SIGMOD Rec.*, 29(4):64–71, 2000.

[18] M. Pöss, B. Smith, L. Kollár, and P. Larson. TPC-DS, taking decision support benchmarking to the next level. In M. J. Franklin, B. Moon, and A. Ailamaki, editors, *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, USA, June 3-6, 2002*, pages 582–587. ACM, 2002.

[19] F. Raab. TPC-C - The standard benchmark for online transaction processing (OLTP). In J. Gray, editor, *The Benchmark Handbook for Database and Transaction Systems (2nd Edition)*. Morgan Kaufmann, 1993.

[20] V. Raman, G. K. Attaluri, R. Barber, N. Chainani, D. Kalmuk, V. KulandaiSamy, J. Leenstra, S. Lightstone, S. Liu, G. M. Lohman, T. Malkemus, R. Müller, I. Pandis, B. Schiefer, D. Sharpe, R. Sidle, A. J. Storm, and L. Zhang. DB2 with BLU acceleration: So much more than just a column store. *Proc. VLDB Endow.*, 6(11):1080–1091, 2013.

[21] A. Raza, P. Chrysogelos, A. G. Anadiotis, and A. Ailamaki. Adaptive HTAP through elastic resource scheduling. In D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, and H. Q. Ngo, editors, *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 2043–2054. ACM, 2020.

[22] P. J. Sadalage and M. Fowler. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Addison-Wesley, Upper Saddle River, NJ, 2013.

[23] Wikipedia contributors. Hybrid transactional/analytical processing — Wikipedia, the free encyclopedia, 2020. [Online; accessed 19-October-2020].