CH3: A Mixed Workload Benchmark for Scalable NoSQL

Mehnaz Tabassum Mahin[1†], Bo-Chun Wang[2], Kamini Jagtiani[2‡], Michael Carey[23], and Keshav Murthy[2]
[1]University of California, Riverside, [2]Couchbase, Inc., [3]University of California, Irvine
[1]mmahi004@ucr.edu

***Abstract*—Database management systems that support hybrid workload (i.e., HTAP or HOAP) first arose in the relational world. Such hybrid data management support in the document database of the NoSQL world is also gaining popularity in both commercial and research arenas. The CH2 benchmark was proposed in 2021 to evaluate such hybrid NoSQL platforms. In addition to the operational and analytical services, full-text search is a key component of NoSQL platforms that provides search engine-like query processing capability on JSON documents. In this paper, we present CH3, a mixed workload benchmark for evaluating scalable NoSQL platforms with OLTP, OLAP, and full-text search (FTS) workloads. Like CH2, the CH3 benchmark borrows from and extends both TPC-C and TPC-H. However, CH3 generates meaningful text contents and includes necessary FTS indexes and relevant FTS queries on these indexes to handle the FTS workload. This paper presents the required extensions from CH2 to address the FTS workload, the detailed design of CH3, and the performance results by running the CH3 benchmark against Couchbase Server (that offers Query, Analytics, and Search services). The results provide insight into the performance of Search Service, the performance isolation among OLTP, OLAP and FTS workloads, and the horizontal scalability of Couchbase Server as well as the effectiveness of CH3 for evaluating a mixed workload performance of such NoSQL platforms.**

***Index Terms*—benchmark, NoSQL, OLTP, OLAP, full-text search**

I. Introduction

Traditionally, operational (OLTP) and analytical (OLAP) processing are categorized as two separate workloads, with each running on their own separate infrastructures. OLTP systems serve transaction-oriented applications and provides high availability and low latency to a large number of users. OLAP systems analyze large amounts of business data for a relatively small number of users. Both systems play vital role to the day-to-day operations of enterprises and organizations. To address the pressing need for timely analytics, and bridge OLTP and OLAP in a single infrastructure, database system architectures with hybrid data management support – referred to as HTAP (Hybrid Transactional/Analytical Processing [1]) or HOAP (Hybrid Operational/Analytical Processing [2]) support – have gained traction in both industry and research. 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.

Traditional relational database systems are designed for strict consistency and data control; thus they tend to fall short of the agility, flexibility, and scalability demands of today's new applications that support millions of interactions with end-users. This has led to the emergence of the new generation of data management systems known as NoSQL systems [3]. Couchbase Server [4] and MongoDB [5] are two examples of document databases, sub-category of such NoSQL systems. 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, thus enabling the players – ranging from application developers to DBAs and data analysts as well – to work with their data in its natural form. In this paper, we focus on NoSQL systems and aim to explore key aspects of NoSQL platforms' support for HOAP [1] and search engine-like full-text search.

In the relational world, TPC-C [6] and TPC-H [7] are two standard benchmarks, where TPC-C is for transactional query processing performance and TPC-H is for analytical processing performance. In 2011, a mixed workload CH-benCHmark [8] was introduced in the DBTest workshop, it measures the performance by combining a transactional workload based on TPC-C and a corresponding TPC-H-equivalent analytical workload. For the NoSQL world, a mixed workload, CH2 [9] was introduced in the TPCTC conference in 2021. It is based on extending and improving the original CH-benCHmark. It evaluates hybrid NoSQL platforms with operational and analytical workloads.

It is to be noted that both TPC-C and TPC-H, and even CH2, mainly focus on numeric, date/time, and simple string fields; and text fields are not queried in these existing benchmarks. However, Full-Text Search (FTS) is an important service of NoSQL platforms that provides a Google-like search capability on JSON documents. A benchmark has yet to be proposed to evaluate FTS workload performance for scalable NoSQL systems. In this paper, we propose a new benchmark for evaluating a mixed workload performance with operational, analytical, and FTS in the document database world.

The remainder of the paper is structured as follows: Section II briefly studies related works on HTAP systems along with existing SQL and NoSQL benchmarks. Section III dis-

†This work was done while Mehnaz Tabassum Mahin was interning at Couchbase, Inc.

‡This work was done while Kamini Jagtiani was at Couchbase, Inc.

[1]We prefer the term HOAP over HTAP in the context of NoSQL, as it seems less tied to strict ACID transactions and columnar, main-memory technology presumptions.

cusses an overview of Couchbase Server and its approach to supporting HOAP and FTS. Section IV describes the CH3 benchmark design details. Section V presents the performance results obtained by running our benchmark on a Couchbase Server cluster with different workload settings. Section VI concludes the paper with a summary of the CH3 benchmark and the results.

II. Related Work

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

A. HTAP (HOAP)

As mentioned in Section I, the relational database world has witnessed an emergence of HTAP capabilities in a number of vendors' systems in recent years as well as growing research interest related to HTAP. Noteworthy HTAP offerings include such systems as HyPer [10] (born in research, but now owned by and used in Tableau) and SAP-HANA [11]. Other significant commercial relational HTAP offerings include DB2 BLU from IBM [12], Oracle's dual-engine main-memory database solution [13], and the real-time analytical processing capabilities now found in Microsoft's SQL Server [14]. As an example on the research side, a recent paper introduced and explored 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 [15]. Snowflake has recently announced Unistore [16] running both transactional and analytical data in a single platform.

Stepping back, one sees that R&D in the relational HTAP world has focused largely on in-memory scenarios for relatively "small" operational databases. Now that multi-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 much or even all of an enterprises' operational business data. As a result, most current HTAP database offerings rely on main-memory database technology. And, as would then be expected, the focus of these offerings is on single-server architectures – i.e., on scaling up rather than scaling out.

In contrast, to the relational world, 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 on Big Data – and flexible, schema-less data. In addition, NoSQL systems and applications have somewhat different transactional consistency needs [3]. Data timeliness is equally important in the NoSQL world, but there is less of a need to focus on reducing or eliminating 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.

B. Benchmarks

Many benchmarks have been developed to evaluate the performance of relational database systems under different application scenarios [17]. The most notable are the TPC-x benchmarks developed by the Transaction Processing Council (TPC). These include TPC-C [6] for transaction processing as well as TPC-H [7] and TPC-DS [18] for decision support and analytics. There has also been a variety of benchmarks proposed and employed in the NoSQL world, including YCSB [19] for key-value store workloads, BigFUN [20] for Big Data management platform operations' performance, MongoDB's adaptation of TPC-C to evaluate NoSQL transactional performance [5], and a philosophically similar NoSQL adaptation [21] of TPC-H to evaluate Big Data analytics performance, to name a few of the NoSQL and Big Data benchmarks.

To evaluate HTAP systems, an especially noteworthy effort was the mixed workload CH-benCHmark [8]. This benchmark resulted from a Dagstuhl workshop attended by a group of database query processing and performance experts drawn from both companies and universities. The CH-benCHmark combines ideas and operations from TPC-C and TPC-H 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 results from applying the benchmark to PostgreSQL with all data in memory and a read-committed transaction isolation level. The CH-benCHmark gained some traction for HTAP use, having been used to assess the performance of a new HTAP system and its scheduler [15].

A first step to assess HOAP for scalable NoSQL systems was reported in [22]. It investigated the performance isolation in Couchbase Server (6.6) by mixing concurrent TPC-C NewOrder transactions with a stream of join/group-by/top-K queries. For the NoSQL world, a mixed workload benchmark, CH2 [9] was more recently introduced in the TPCTC conference. CH2 is based on extending and improving the original CH-benCHmark on Couchbase Server (7.0) for NoSQL systems and borrows and adapts both TPC-C and TPC-H ideas.

None of these existing benchmarks consider the full-text search (FTS) workload which is, now-a-days, a vital service of document databases. To the best of our knowledge, our paper presents the first mixed workload benchmark proposed to evaluate HOAP and FTS for scalable NoSQL systems.

III. Couchbase Server

Couchbase Server is a highly scalable document-oriented database management system [4]. 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 two high performance complementary query engines [23] for executing declarative SQL-like N1QL[2] queries. It also includes support for full-text search.

Figure 1 lists Couchbase Server's major components. Architecturally, the system is organized as a set of services that are deployed and managed as a whole on a Couchbase Server

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

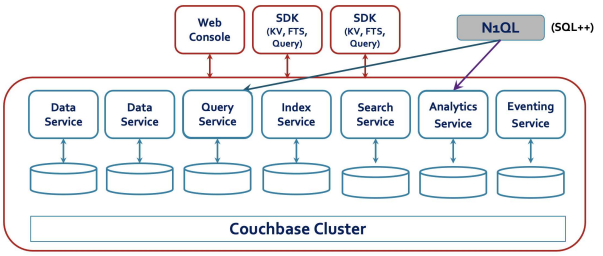


Fig. 1. Major Couchbase Server Components

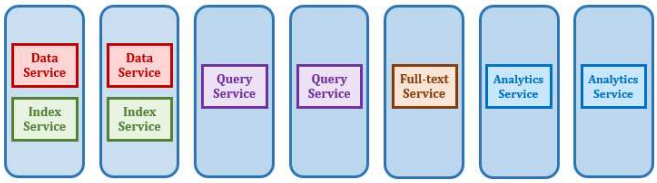


Fig. 2. Multi-Dimensional Scaling (MDS)

cluster. Physically, a cluster is a group of nodes operating in a peer-to-peer topology, and the services on each node can be managed as required. Nodes can be added or removed through a rebalance process that redistributes the data across the cluster's nodes. This can increase or decrease the CPU, memory, disk, or network capacity of a cluster. The ability to dynamically scale the cluster and map services to sets of nodes is referred to as Multi-Dimensional Scaling (MDS). Figure 2 shows how MDS might enable a small Couchbase Server cluster to have 2 nodes shared by its Data and Index Services, 2 nodes for the Query Service, 1 node for the Full-Text Search Service, and 2 nodes for the Analytics Service.

A key aspect of Couchbase Server's architecture is the manner in which data changes are communicated across services. An internal Database Change Protocol (DCP) continuously notifies all services of the changes to documents managed by the Data Service in Figure 1.

The Data Service is the foundation for document management. It provides caching, persistence, and inter-node replication. The document data model is JSON. Documents are stored in containers called buckets. A "bucket" contains related documents, similar to a database in a relational database (RDBMS). There is no explicitly defined schema, so the "schema" for documents 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 by deploying new application code that stores new JSON data without having to also make and deploy corresponding changes to a static schema. As of Couchbase Server 7.0 or higher, documents within a bucket reside in collections (similar to RDBMS tables) that can be grouped together logically using scopes (similar to RDBMS schemas).

The Indexing, Search, and Query Services coordinate via DCP to provide document database management functionality that supports high volumes of low-latency queries and up-

Fig. 3. Analytics Service in Couchbase Server

dates for JSON documents. The Indexing Service provides global secondary indexing for the data managed by the Data Service. The Search Service provides richer text indexing to support a wide range of possible search engine-like querying, i.e., full-text search (FTS); plus a set of APIs for search-oriented applications that prefer to interact with this service directly. The Query Service ties this all together by exposing Couchbase Server's database functionality through N1QL [24], [25], a declarative, SQL-based query language that relaxes the rigid 1NF and strongly-typed schema demands of the relational SQL standard. As of Couchbase Server 7.0 or higher, N1QL supports SQL-style, multi-document, multi-statement transactions using a combination of optimistic and pessimistic concurrency control. A series of N1QL DML statements can be grouped into an atomic transaction whose effects span the Query, Indexing, Search and Data Services. We discuss details of Couchbase Server's Search Service in Section III-A.

The Analytics Service complements the Query Service by supporting more expensive ad-hoc analytical queries (e.g., large joins and aggregations) over JSON document collections. Figures 3(a) and 3(b) show its role in Couchbase Server. The Data and Query Services provide low-latency key-value-based and query-based access to their data. Their design point is operational; they support many users making well-defined, programmatic requests that tend to be small and inexpensive. In contrast, the Analytics Service focuses on ad hoc and analytical requests; it has fewer users posing larger, more expensive N1QL queries against a real-time shadow copy of the same JSON data. The Query service has a largely point-to-point/RPC-based query execution model; the Analytics Service employs partitioned parallelism under the hood, using parallel query processing to bring all of the resources of the Analytics nodes to bear on each query [23].

The Eventing Service offers an Event-Condition-Action based framework that provides near real-time handling of data changes in the Couchbase cluster.

So what about HOAP? As Figures 3(a) and 3(b) try to indicate, operational data in Couchbase Server is available for analysis as soon as it is created; analysts always see fresh application data thanks to DCP. They can immediately pose questions about operational data, in its natural data model, reducing the time to insight from days or hours to seconds. There are several differences between this approach and HTAP in the relational world. One is scale: The Analytics Service can be scaled out horizontally on a shared-nothing cluster [23], and it can be scaled independently (Figure 2). It maintains a

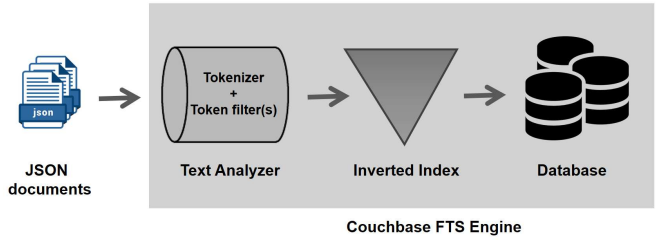


Fig. 4. Search Service in Couchbase Server

real-time shadow copy of operational data that an enterprise wants to analyze; the copy is because Analytics is deployed on its own nodes with their own storage 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 and access frequency would not warrant the cost of a memory-resident solution, but whose aggregated content can still be invaluable for decision-making.

A. Couchbase Search Service

The Search Service in Couchbase Server supports language-aware searching by analyzing textual and other contents of JSON documents and building multi-purposed FTS indexes within a Couchbase bucket. The indexes created and used by the Search Service are designed to provide scores to the search results based on their relevancy and to handle FTS workloads very efficiently. These indexes are entirely separate from and different to those of the Indexing Service.

Figure 4 shows the major components of the Couchbase Server Search Service. The FTS engine handles text indexing for JSON documents. It analyzes the ingested data from the server using the text analyzer [3], builds an *inverted index* for the analyzed contents, and stores into the FTS database. The text analyzer, consisting of a tokenizer and token filters, breaks down the raw text into a list of words, referred as *tokens*. These tokens are more suitable for indexing in the database and searching documents. Depending on the definition of the tokenizer and token filters, different tokens are generated. Consider a sample text: "Jurassic Park". The *whitespace* token filter breaks the sample text into tokens when a whitespace is found, and outputs "jurassic", "park" as tokens. In contrast, the *edge-n-gram* token filter forms n-grams of specified length. If the minimum length is 3 and maximum length is 5, then it generates "jur", "jura", "juras", "sic", "ssic", "assic" "par", "park", "ark" as tokens. Other commonly used token filters include *to_lower, stop_en, stemmer*, etc. The *to_lower* token filter converts all characters to lowercase; while the *stop_en* token filter removes unnecessary tokens for FTS, e.g., "is", "and", "or", "the", etc. The *stemmer* token filter transforms tokens following the Porter Stemming Algorithm [26]; for example, it transforms both "feels" and "feeling" tokens as "feel".

In Couchbase Server, there are some pre-constructed analyzers, including keyword, simple, standard, etc. The *keyword* analyzer creates a single token representing the entire input; while the *standard* and *English* analyzers apply the Unicode tokenizer and *to_lower* and *stop_en* token filters. Along with these token filters, *English* analyzer uses the *stemmer* token filter. As a result, the *standard* analyzer generates "jurassic", "park" as tokens while the *English* analyzer generates "jurass", "park" as tokens for the above mentioned sample text. Apart from these predefined analyzers and filters, one can create custom token filters and analyzers in Couchbase Server. To run a FTS query on a FTS index, the analyzer specified for a query needs to be the same as specified in the FTS index. If no analyzer is specified in the query, the analyzer used for the corresponding FTS index is used. Note that the non-analytic FTS queries do not support any text analyzers on the query texts.

IV. CH3 Benchmark Design

When we undertook the effort reported here, we aimed to explore several key aspects of NoSQL platforms' support for HOAP together with search engine-like full-text search. To investigate the multi-dimensional scaling and measure the performance of a 3-part mixed workload with operational (OLTP), analytical (OLAP) and Full-Text Search (FTS) queries, here we introduce a new benchmark, CH3. We extended the CH2 framework [9] for CH3. In this extended benchmark, our goal was to more fully explore the performance of NoSQL platforms, including (1) workload isolation among OLTP, OLAP and FTS, (2) scaling of sub-workloads, and (3) performance of the full-text search service. These were of interest because most NoSQL systems are designed to scale out horizontally on shared-nothing clusters and their initial design points for query processing have been OLTP-oriented, i.e., they are generally built to support high-concurrency/low-latency operational workloads as opposed to more complex data analytics and search engine-like full-text search workloads.

So far we have investigated the CH3 benchmark on a Couchbase Server cluster. However, CH3 is not specific to Couchbase Server; rather it can also be implemented on other NoSQL platforms that support query, analytics and full-text search services. The rest of this section discusses details of the CH3 benchmark.

A. Benchmark Schema

The CH3 schema is adapted from the CH2 schema, which in turn is a modified version of the original CH-benCHmark [8] schema adapted for the NoSQL world. Figure 5 shows the CH2/CH3 schema. It summarizes the 9 tables and relationships of the standard relational TPC-C schema. This schema models businesses which "must manage, sell, or distribute products or services" [6] and it follows a continuous scaling model.

The Order and Order-Line tables of TPC-C were combined in CH2 by having Order-Line inlined as arrays inside of the

[3]The details on the text analyzers in Couchbase Server can be found at https://docs.couchbase.com/server/current/fts/fts-index-analyzers.html

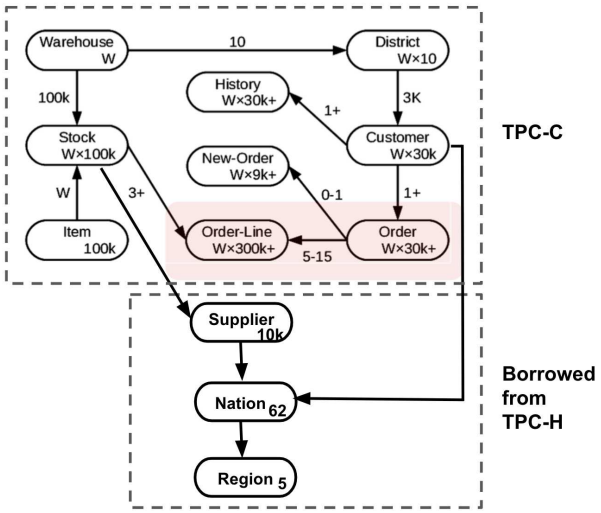


Fig. 5. CH2/CH3 Schema.

TABLE I
CH2/CH3 COLLECTIONS [9] AND EXAMPLE SIZES WITH W = 1,000 WAREHOUSES.

Collection	Collection Size (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 (Order-Line)	30,000,000 (300,000,000)
Item	100,000
Supplier	10,000
Nation	62
Region	5

Orders collection, and it retains the other 7 tables from TPC-C. No other nesting changes were made to the TPC-C schema, as doing so would involve over-nesting and would be a poor database design for such use cases[5], [21]. Like the CH-benCHmark, in addition to adopting the nested order modification and TPC-C's scaling rules, CH2 borrows 3 TPC-H tables with some modifications as additional CH2 collections to support a TPC-H-equivalent analytical workload. Following CH, CH2 borrows Supplier and Region, both unchanged, from the TPC-H schema, along with a slightly modified version of Nation. The modifications in collections from TPC-C are highlighted and the collections borrowed from TPC-H are indicated in Figure 5.

In the CH3 benchmark, we use all of the collections from CH2 for the operational and analytical workloads. For the FTS workload, we mainly focused our extensions on the collections with meaningful text fields, and so, considered all collections except Warehouse, New-Order, Nation and Region. Note that, like CH and CH2, the CH3 benchmark database size is scalable based on the number of warehouses (W) and it adopts the TPC-C scaling rules.

Table I lists the CH2/CH3 collections and gives an example of their scaling by listing their 1,000-warehouse cardinalities. Orders are nested with an average of 10 Order-Line items in each. The line separates the modified TPC-C collections (top) from the three CH (and CH2/CH3) additions (bottom).

B. Benchmark Data

While designing the CH3 benchmark, we first encountered a major problem – the text fields in CH2 are just space-filling gibberish (as in TPC-C, TPC-H, and CH). To execute any FTS queries, we needed human-readable and meaningful texts so that we can execute meaningful FTS queries. One way to address this problem would have been to use a real-world dataset with a TPC-C-like schema. However, we found it difficult to find a real-world dataset which follows the TPC-C schema. As a next idea, we considered using FakerJS [4] for generating some fake datasets, especially for the text fields. Though FakerJS could serve our purpose to some extent, it fails to generate meaningful item names and data, and there is no correlation among city, state and zip code values. Instead, we ended up borrowing field values from an eCommerce inventory dataset from Flipkart [27] and a worldwide zip code population dataset alongside FakerJS. Overall, the CH3 database generator is a modified version of the CH2 data generator which now supports OLTP, OLAP and FTS queries.

Table II indicates the major changes in the CH3 database generator from the CH2 data generator. Note that we did not modify `c_last` of Customer in order to maintain compliance with the database population requirement of TPC-C (Clause 4.3.2.3). Also, the text fields for the collections, not mentioned in Table II, are directly adapted from CH2 in CH3. Figure 6 shows some examples of how the CH3 benchmark documents in the Item and District collections differ from the corresponding CH2 documents. One can see that the numeric fields in the example documents of the Item and District collections are same in both CH2 and CH3, and the text fields differ in CH3 from CH2 as mentioned in the table. The CH2 data generator generates `i_data` and `i_name` fields as space-filling and meaningless texts while the CH3 data generator borrows values from the eCommerce Flipkart dataset for these fields and provides some meaningful texts. Similarly, the text values of the District document in CH2 are also meaningless. In contrast, in CH3, values are borrowed from the Zip code population dataset for `d_city, d_state, d_zip` fields, and FakerJS is used to generate `d_street_1` and `d_street_2` field values of the District document. These meaningful texts allow us to execute meaningful FTS queries on the CH3 documents.

C. Benchmark Indexes

In Couchbase Server, every Full-Text Search is performed via a user-created Full Text Index that contains the targets on which searches are to be performed. These targets are values derived from the textual and other contents of documents within a specified bucket or collections within a scope.

[4]https://github.com/faker-js/faker

TABLE II
CH3 DATABASE GENERATOR USING VALUES FROM REAL DATASETS AND FAKERJS PYTHON LIBRARY.

Source of field values	Collection(s)	Field(s)	Comments on generated data

Zip code population dataset	Warehouse
	District
	Customer
		`w_city, w_state, w_zip`
		`d_city, d_state, d_zip`
		`c_city, c_state, c_zip`
			The `_zip` values do not comply with the database population requirement of TPC-C (Clause 4.3.2.7).

eCommerce Inventory dataset (Flipkart)	Item	`i_name, i_data`	Both of the fields are truncated to meet the length requirements of TPC-C and support the adaptation of TPC-C queries.

FakerJS python library	Warehouse
	District
	Customer
		`w_street_1, w_street_2`
		`d_street_1, d_street_2`
		`c_street_1, c_street_2`
			These fields are not correlated with the `_city, _state, _zip` values.

	Customer	`c_data, c_first, c_phone`	The `c_data` field is a paragraph containing text about the profile creation date, username, job_title, company, email and website url of each customer.
	History Stock	`h_data` `s_data`	These values meet the length requirements of TPC-C.
	Supplier	`su_address, su_phone`	

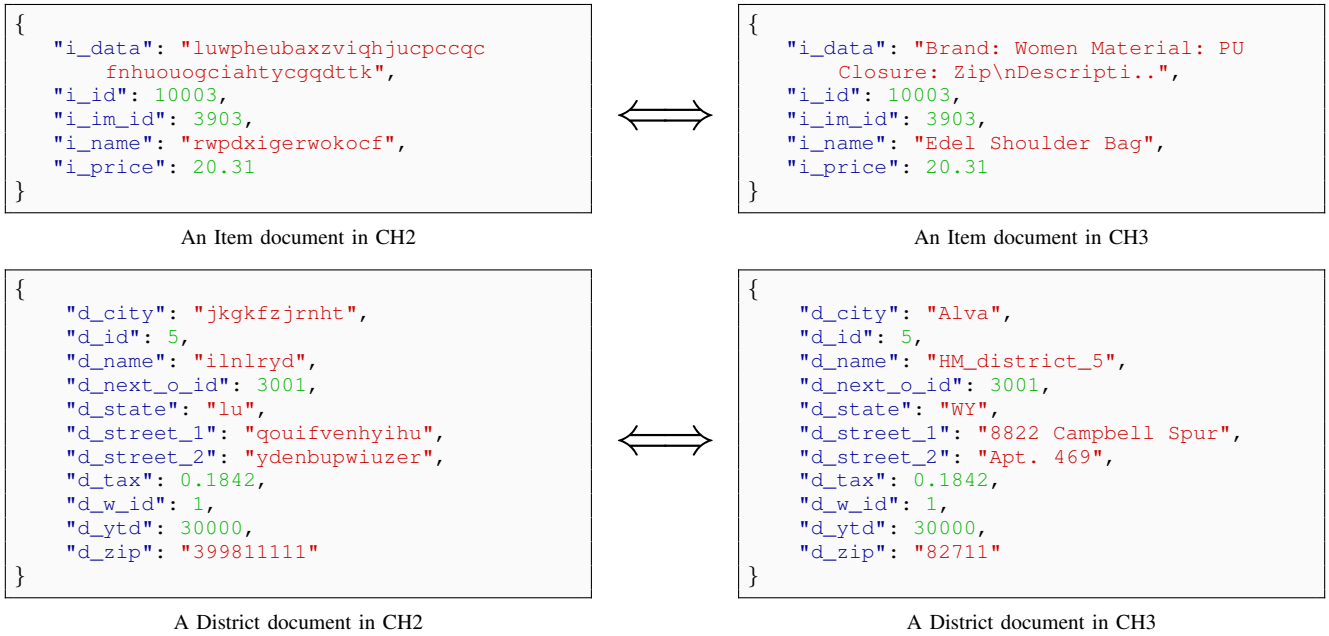


Fig. 6. Examples of showing differences between CH2 and CH3 documents.

Documents can also be grouped by the user across multiple buckets/scopes.

The CH3 data are stored in a single scope *ch3* in a bucket called *bench* in the Data Service, so we focused on creating Full Text Indexes in the Search Service within that scope. We used the Full-Text Search REST API for creating these indexes where a JSON document (the body of the REST request) contains the definition of an index. We created 3 indexes on single collections and 3 indexes across multiple collections within the bucket/scope [5]: (1) *customerFTSI* on Customer, (2) *itemFTSI* on Item, (3) *ordersFTSI* on Orders, (4) *ngramFTSI* on History and Stock, (5) *nonAnalyticFTSI* on Customer, Stock and Supplier, and (6) *multiCollectionFTSI* on Customer, District and Orders collections.

For CH3, we considered different text analyzers like keyword, English, edge-n-gram analyzers, etc. in different FTS indexes. The *customerFTSI* index uses keyword and English text analyzers on different fields to index the documents. Specifically we applied the keyword analyzer on the `c_first` field in the Customer collection so that it forces exact matches and preserves characters such as spaces during indexing the documents. The other indexed fields in *customerFTSI* follow the English analyzer. Similarly, the *nonAnalyticFTSI* index uses keyword and English text analyzers on different fields. We considered the default standard analyzer for the indexed fields in *itemFTSI, ordersFTSI*, and *multiCollectionFTSI* in-

[5]The FTS index definitions for the CH3 benchmark are available at https://github.com/couchbaselabs/ch3

dexes. In contrast, the *ngramFTSI* index uses a customized token analyzer using an edge-n-gram token filter to index the documents.

D. Benchmark Queries

To design the mixed workload of CH3, we included all the operational and analytical queries from CH2 without any modifications against the CH3 benchmark data. These queries are expressed in N1QL (i.e., SQL++), a SQL-like query language that considers nested data. In addition to the CH2 queries, we have included 20 FTS queries. We pose the FTS queries in JSON format so as to execute them via the Search Service of Couchbase Server. When running FTS queries in N1QL format, the queries are sent to the Query Service, rerouted to the Search Service to fetch the results, and the results are returned from the Query Service to the client node. To maintain better isolation, we decided to send the FTS queries in JSON format so that operational and FTS queries can be run separately by the Query Service and Search Service components, respectively.

In Couchbase Server, the Full-Text Search Service allows us a wide range of query options, including match queries, match-phrase queries, range queries, non-analytic queries, compound queries, etc. Match and match-phrase FTS queries can analyze their input texts and perform fuzzy and/or prefix matches. The fuzziness of a query can be specified so that the scope of the matches can be constrained to a particular level of exactitude. In contrast, non-analytic FTS queries do not support analysis on their query texts and will return only exact matches. Fuzziness, regular expressions, wildcards, etc., can be specified for a non-analytic query. Range queries can find documents containing a value in the specified field within the specified range, where the specified range can be either of a date, numeric or term range. Compound queries are designed to accept multiple queries simultaneously, and return either the conjunction of results from the result-sets of all child queries, or a disjunction.

We designed the 20 FTS queries in CH3 so that together they cover all of supported FTS query types in Couchbase Server. We categorized these FTS queries into *Simple, Advanced*, and *Non-analytic* queries. Among the 20 FTS queries, 6 are simple queries, 8 are advanced queries and 6 are non-analytic queries. *Simple queries* are simple match and match-phrase queries, whereas compound queries and relatively complex match queries are categorized as *advanced queries*.

Table III shows examples of a simple FTS query and an advanced FTS query in JSON format. The name of collections and corresponding FTS indexes on which we can execute FTS queries are also mentioned along with the queries in the table. The left query is a simple query that is basically a range query searching for a range of dates from the Orders collection, and the corresponding FTS index named *ordersFTSI* indexes the `o_entry_d` field of the Orders collection. The right query is an advanced query that finds documents with matching terms from two fields of two collections, i.e., documents that match the given terms in the `h_data` field of the History and the `s_data` field of the Stock collection. The corresponding FTS index named *ngramFTSI* indexes both `h_data` and `s_data` fields of the History and Stock collections based on a customized token analyzer using an edge-n-gram token filter of min length 3 and max length 7.

Note that as the number of results obtained for a FTS query can be large at times, we can limit how many results to fetch starting at which offset by specifying `size` and `from` parameters in the FTS query. As a result, in response to a FTS query, `size + from` number of results are fetched and `size` number of results are returned starting at offset `from`.

V. CH3 Benchmark Results

In this section, we report the performance results from implementing and running our mixed workload CH3 benchmark on a 6-node Couchbase Server cluster. Our primary focus here is to use CH3 to explore several key aspects of NoSQL platforms, including (1) performance isolation among OLTP, OLAP and FTS workloads, (2) the scalability of their architectures with mixed workloads, and (3) full-text search service performance.

A. Benchmark Implementation

To implement the CH3 benchmark's mixed workload, we started with the CH2 adaptation of CMU's py-tpcc benchmark system, the same package recently used by MongoDB [5]. We then modified CH2's data generator following the earlier description, added FTS indexes and queries in JSON, and finally added a CH3 driver for Couchbase Server to meet our mixed workload requirements [6].

The CH3 data resides in a scope called *ch3* in a bucket called *bench* in the Data Service (in JSON document form). The required indexes to support the operational queries and updates were created in the *bench* bucket in the Data Service. In contrast, the required FTS indexes to support the FTS workload were created in the *bench* bucket in the Search Service. For these experiments, we generated a 1,000-warehouse instance of CH3. The cardinalities of the CH3 collections are thus consistent with the example numbers shown in Table I.

Each operational, analytical, or FTS user is simulated by a stream running on a client node of the configured Couchbase Server cluster. Each stream consistently sends query requests to the system. 0-128 streams send TPC-C operations to the Query Service, 0-128 streams send FTS query requests to the Search Service, with 0 or 1 stream sending analytical queries to the Analytics Service. These stream counts simulate a typical business model with more front-end users (either running OLTP or FTS queries) than data analysts.

B. Benchmark Configuration(s)

As mentioned earlier, we ran our CH3 benchmark implementation on a cluster consisting of 6 nodes. Hardware-wise, the cluster was comprised of 5 nodes, each with 24 vCPUs, 64GB of memory, and up to 10 Gbps of network bandwidth,

[6]The software artifacts associated with this paper's benchmark are available at https://github.com/couchbaselabs/ch3

TABLE III
A SIMPLE FTS QUERY (LEFT) AND AN ADVANCED FTS QUERY (RIGHT)

Collection: Orders FTS Index: ordersFTSI	Collections: History, Stock FTS Index: ngramFTSI

<pre>{
  "explain": false,
  "fields": [
    "*"
  ],
  "highlight": {},
  "query": {
    "start": "2015-01-01",
    "end": "2016-10-30",
    "field": "o_entry_d"
  },
  "size": 5,
  "from": 0
}</pre>
	<pre>{
	 "explain": false,
	 "fields": [
	    "*"
	],
	 "highlight": {},
	 "query": {
	   "disjuncts": [
	     {
	       "match": "Expert opinion",
	       "field": "h_data"
	     },
	     {
	       "match": "international policy",
	       "field": "s_data"
	     }
	   ]
	 },
	 "size": 5,
	 "from": 0
	}</pre>

forming the Couchbase Server cluster. Moreover, there is one client node with 48 vCPUs, 64GB of memory, one 480GB SSD, and up to 10 Gbps of network bandwidth that was used to run the client workload driver. The nodes running a Data, Index, Query, and Search Service utilized one 1TB SSD drive, while the nodes running the Analytics Service utilized two 1TB SSDs drives uniformly for enhanced query parallelism.

We show the results with six different workload settings: (1) only FTS streams, (2) only TPC-C streams, (3) CH2 streams, i.e. both TPC-C and TPC-H, (4) FTS and TPC-C streams, (5) FTS and TPC-H streams, (6) FTS and CH2 streams. For these settings, we vary the number of both FTS and operational streams equally as 1, 2, 4, 16, 32, 64, 96, 128, and consider 0-1 number of analytical stream.

C. Initial Benchmark Results

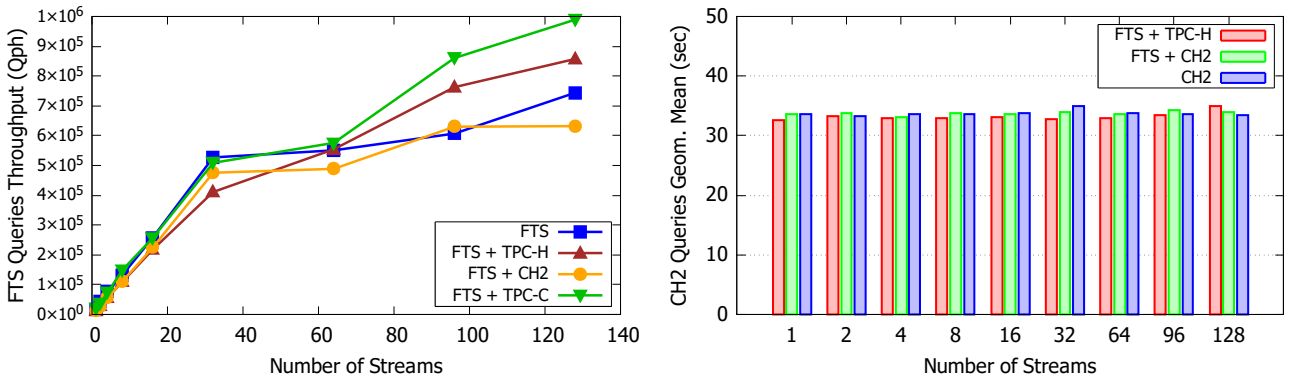
Our initial goal is to investigate Couchbase Server's performance isolation for CH3's mixed workload, FTS performance and scalability of workloads. To measure the performance with CH3 workload, operational, analytical, and FTS clients were run concurrently until the analytical client running the 22 analytical queries completed one full loop. For a FTS client, we executed 25% simple, 35% non-analytic, and 40% advanced FTS queries as long as the analytical client completed the loop. In case of 0 analytical stream, operational and FTS clients were run concurrently for a certain duration.

Figure 7 shows the results that reveal about Couchbase Server's FTS and analytical query performance. Figure 7(a) shows the throughput for the FTS queries in Queries per hour (Qph) by varying the number of operational and FTS streams equally from 1, 2 to 128 with our different workload settings. We can see that the FTS queries' throughput exhibit a textbook performance increase as the number of streams is increased. There are two key observations in this graph. The first one is the expected scalability of FTS workload for a variable number of streams. The second is the FTS workload isolation with or without running operational and analytical queries.

Figure 7(b) shows the geometric mean of the 22 analytical queries' average response times in seconds vs. the number of streams. Here, we vary the number of operational and FTS streams equally from 1, 2 to 128 for these results. From this set of results, we can observe the analytical workload performance isolation which is not affected while the number of operational and FTS streams is increased on the x-axis.

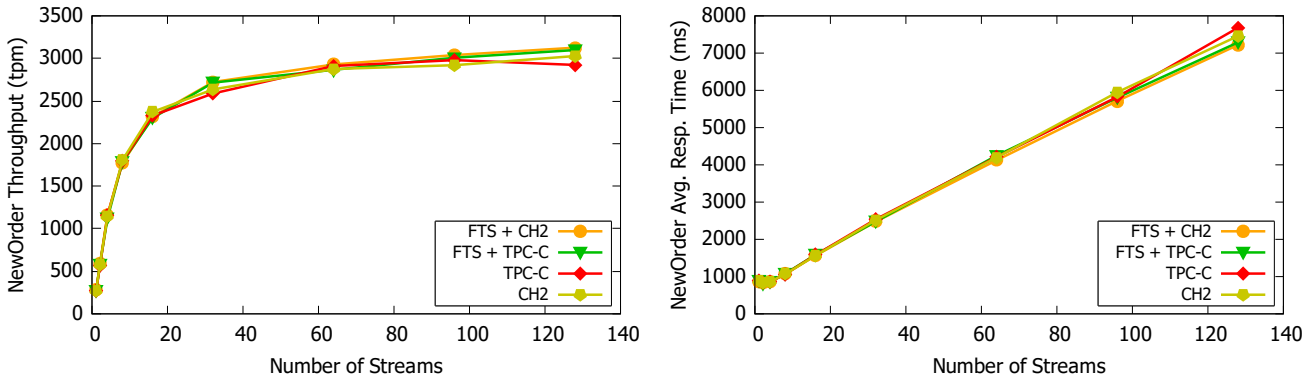
Now, we discuss the NewOrder transaction results with our different workload settings. Figure 8(a) shows the NewOrder throughput in transactions per minute (tpm) running a mixed workload with 0-1 analytical client and varying the number of operational and FTS streams equally from 1, 2 to 128. We can observe the throughput increasing linearly at the beginning and then reaching a plateau when the resources are saturated. Also, the throughput exhibits the same patterns and trends for all different workload settings. Figure 8(b) shows the corresponding average response times in milliseconds for the NewOrder transactions. The response times are linearly proportional to the number of streams. So, we can observe that these two graphs are pairwise identical and also depict the performance isolation of operational workload with different workload settings.

The results discussed in this section clearly show the *effective performance isolation* of Couchbase Server cluster components with a mixed workload of operational, analytical and FTS streams. Also, one can notice the *scalability* of its architecture when faced with a need to support more



(a) (b)

Fig. 7. (a) FTS Queries Throughput (Qph) and (b) CH2 Queries Geometric Mean (sec)., varying the number of streams.



(a) (b)

Fig. 8. NewOrder Throughput (tpm) and (b) NewOrder Average Response Time (ms), varying the number of streams.

operational or FTS users.

VI. Conclusion

Database management systems with hybrid workload support – HTAP or HOAP – first appeared in the relational world, where they are often linked to server technology trends such as columnar storage and memory-rich, many-core, scale-up server technology. Such hybrid data management support in the document database of the NoSQL world is also being attracted in both industry and research sectors. In addition to the operational and analytical services, full-text search is a key component of NoSQL platforms that provides Google-like search capability on JSON documents. In this paper, we introduced CH3, a benchmark for evaluating scalable NoSQL platforms with a mixed workload – operational (OLTP), analytical (OLAP), and full-text search (FTS). In this effort, we designed CH3 in such a way that it supports meaningful FTS workload as well as OLTP and OLAP workloads. Like CH2, the CH3 benchmark borrows from and extends both TPC-C and TPC-H. However, CH3 generates meaningful text contents and includes necessary FTS indexes and relevant FTS queries on these indexes to handle the FTS workload. We studied the performance of a scalable NoSQL platform, Couchbase Server, that offers Query, Analytics, and Search services, by running the CH3 benchmark. The performance results provide insight into the performance of Search Service, the performance isolation among OLTP, OLAP and FTS workloads, and the horizontal scalability of Couchbase Server. It also exhibits the importance of CH3 for evaluating a mixed workload performance of NoSQL platforms.

Acknowledgments

The authors wish to thank Abhinav Dangeti of Couchbase Full-Text Search team who has provided technical assistance related to the new FTS features. They would also like to thank Peter Reale of Couchbase, Inc. for assisting with the Zip code population dataset.

References

[1] Wikipedia contributors, "Hybrid transactional/analytical processing — Wikipedia, the free encyclopedia," 2020, [Online; accessed 19-October-2020]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Hybrid_transactional/analytical_processing&oldid=981969658

[2] 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].

[3] P. J. Sadalage and M. Fowler, *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Upper Saddle River, NJ: Addison-Wesley, 2013.

[4] D. Borkar *et al.*, "Have your data and query it too: From key-value caching to big data management," in *Proc. ACM SIGMOD Conf.* ACM, 2016, pp. 239–251.

[5] A. Kamsky, "Adapting TPC-C benchmark to measure performance of multi-document transactions in MongoDB," *PVLDB*, vol. 12, no. 12, pp. 2254–2262, 2019.

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

[7] M. Pöss and C. Floyd, "New TPC benchmarks for decision support and web commerce," *SIGMOD Record*, vol. 29, no. 4, pp. 64–71, 2000.

[8] R. L. Cole *et al.*, "The mixed workload CH-benCHmark," in *Proc. Fourth Int'l. Workshop on Testing Database Systems, DBTest*. ACM, 2011, pp. 1–6.

[9] M. Carey, D. Lychagin, M. Muralikrishna, V. Sarathy, and T. Westmann, "CH2: A Hybrid Operational/Analytical Processing Benchmark for NoSQL," in *Performance Evaluation and Benchmarking: 13th TPC Technology Conference, TPCTC 2021, Copenhagen, Denmark, August 20, 2021, Revised Selected Papers*. Springer-Verlag, 2021, p. 62–80.

[10] A. Kemper and T. Neumann, "Hyper: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots," in *2011 IEEE 27th Int'l. Conf. on Data Engineering*, 2011, pp. 195–206.

[11] N. May, A. Böhm, and W. Lehner, "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.

[12] V. Raman *et al.*, "DB2 with BLU acceleration: So much more than just a column store," *PVLDB*, vol. 6, no. 11, pp. 1080–1091, 2013.

[13] T. Lahiri *et al.*, "Oracle database in-memory: A dual format in-memory database," in *2015 IEEE 31st Int'l. Conf. on Data Engineering*, 2015, pp. 1253–1258.

[14] P. Larson *et al.*, "Real-time analytical processing with SQL server," *PVLDB*, vol. 8, no. 12, pp. 1740–1751, 2015.

[15] A. Raza *et al.*, "Adaptive HTAP through elastic resource scheduling," in *Proc. ACM SIGMOD Conf.* ACM, 2020, pp. 2043–2054.

[16] (2022) Snowflake Unistore. [Online]. Available: https://www.snowflake.com/en/data-cloud/workloads/unistore/

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

[18] M. Pöss *et al.*, "TPC-DS, taking decision support benchmarking to the next level," in *Proc. ACM SIGMOD Conf.* ACM, 2002, pp. 582–587.

[19] B. F. Cooper *et al.*, "Benchmarking cloud serving systems with YCSB," in *Proc. 1st ACM Symp. on Cloud Computing, SoCC 2010, Indianapolis, Indiana, USA, June 10-11, 2010*. ACM, 2010, pp. 143–154.

[20] P. Pirzadeh, M. Carey, and T. Westmann, "BigFUN: A performance study of big data management system functionality," in *2015 IEEE Int'l. Conf. on Big Data*, 2015, pp. 507–514.

[21] P. Pirzadeh, M. Carey, and T. Westmann, "A performance study of big data analytics platforms," in *2017 IEEE Int'l. Conf. on Big Data*, 2017, pp. 2911–2920.

[22] Y. Tian, M. Carey, and I. Maxon, "Benchmarking HOAP for scalable document data management: A first step," in *2020 IEEE Int'l. Conf. on Big Data*, 2020, pp. 2833–2842.

[23] M. A. Hubail *et al.*, "Couchbase Analytics: NoETL for scalable NoSQL data analysis," *PVLDB*, vol. 12, no. 12, pp. 2275–2286, 2019.

[24] D. Chamberlin, *SQL++ for SQL Users: A Tutorial*. Couchbase, Inc. (Available via Amazon.com.), 2018.

[25] D. Chamberlin, "Comparing Two SQL-Based Approaches for Querying JSON: SQL++ and SQL:2016," White Paper, Couchbase, Inc., 2019.

[26] (2006) Porter Stemming Algorithm. [Online]. Available: https://tartarus.org/martin/PorterStemmer/

[27] (2017) Flipkart eCommerce Inventory Dataset. [Online]. Available: https://data.world/promptcloud/product-details-on-flipkart-com