

An Embedded Database System for YANG



Han Bing¹, Gao Jirun¹, Zhang Kaiqi^{1,2}, Wang Hongzhi¹,
Chen Hedui³

(1. Harbin Institute of Technology, Harbin 150001, China;

2. Chongqing Research Institute of Harbin Institute of Technology,
Chongqing 401135, China;

3. ZTE Corporation, Shenzhen 518057, China)

DOI: 10.12142/ZTECOM.202603010

<https://kns.cnki.net/kcms/detail/34.1294.TN.20260826.1320.002.html>,
published online August 27, 2026

Manuscript received: 2024-09-19

Abstract: Yet Another Next Generation (YANG) is a data modeling language which is primarily designed to define the hierarchical structure of data used in network configuration management protocols such as NETCONF. However, traditional relational databases struggle to maintain optimal index performance for YANG data, and it is challenging to ensure data validity and transaction characteristics, particularly in resource-constrained environments such as network base stations. In this paper, we introduce the design and implementation of DB4YANG, an embedded database optimized for minimal resource usage while effectively handling and storing YANG model data. DB4YANG supports fast read and write operations on YANG data and ensures data integrity through continuous synchronization with optimistic locking. This approach mitigates the performance drawbacks that arise from non-native processing and format conversions commonly found in relational databases. Furthermore, this paper delves into transaction processing techniques tailored for YANG model data. These techniques are designed to ensure the reliability and consistency of data operations and strictly adhere to the Atomicity, Consistency, Isolation, and Durability (ACID) principles to preserve data integrity. We conducted several experiments comparing DB4YANG with an existing database, and the results demonstrate the effectiveness and efficiency of the proposed database.

Keywords: YANG; embedded database; high-performance index; data consistency

Citation (Format 1): Han B, Gao J R, Zhang K Q, et al. An embedded database system for YANG [J]. *ZTE Communications*, 2026, 24(3): 82 – 91. DOI: 10.12142/ZTECOM.202603010

Citation (Format 2): B. Han, J. R. Gao, K. Q. Zhang, et al., “An embedded database system for YANG,” *ZTE Communications*, vol. 24, no. 3, pp. 82 – 91, Sept. 2026. doi: 10.12142/ZTECOM.202603010.

1 Introduction

In recent years, the global expansion of cloud computing and the surge in business demands have accelerated network technologies and software-defined networking (SDN)^[1]. The essence of SDN is the decoupling of the control plane from the data plane, which was originally achieved via OpenFlow. While OpenFlow is no longer essential, the idea of separating control from forwarding remains central to SDN^[2]. The programmability of networks has progressively evolved into a crucial criterion for evaluating SDN frameworks.

Traditionally, programming network devices means using command line interfaces (CLI) and the simple network management protocol (SNMP). Network operators build scripting and management tools on top of these protocols to achieve some level of network programmability, which is useful for automating numerous tasks. Some devices allow for configuration through a web interface using extensible markup lan-

guage (XML) for bulk settings. However, despite its ease of use, CLI is not efficient or easy to read. It is primarily designed for human-machine interaction rather than the programmatic needs of networks. CLI proves insufficient for large-scale network automation due to its limited programmability and the complexities involved in development.

SNMP was introduced to simplify device configuration. It is a network management protocol comprising application-layer protocols, database schemas, and data objects for network supervision^[3]. SNMP is widely used for monitoring, but it suffers from several limitations. First, it lacks readability. It also has security vulnerabilities and lacks backup and rollback options. Moreover, its write capabilities are limited, which means it is primarily suitable for monitoring and basic configuration checks rather than in-depth configuration management. These limitations hinder its utility in network configuration tasks.

To address the intricacies of device configuration management, the IETF introduced standards including NETCONF, Yet Another Next Generation (YANG), and RESTCONF. NETCONF is an XML-based protocol for network management. It includes a suite of functionalities such as configuration, monitoring, fault management, secure authentication,

*The corresponding author is Zhang Kaiqi.

This work was supported by Natural Science Foundation of Chongqing, China, under Grant No. CSTB2023NSCQ-MSX0673 and ZTE Industry-University-Institute Cooperation Funds under Grant No. 2023ZTE09-02.

and access control. These features have contributed to its widespread industry adoption^[4]. YANG, defined in RFC 6020 (2010), is a modeling language for standardizing NETCONF's data models and operations. The integration of YANG models within NETCONF messages creates a universal network programmability interface. YANG-based network programmability has been widely adopted since 2016, simplifying network administration and reducing operational costs.

YANG provides a concise yet comprehensive means of defining data structures, yielding lightweight data descriptions. Network devices use remote procedure calls (RPCs) that are translated into XML messages, conforming to YANG model rules, to enable seamless end-to-end communication. However, with the advent of technologies such as 5G, managing the ever-growing volume of YANG model data at a large scale presents significant challenges.

Traditional relational databases have difficulty maintaining optimal index performance for YANG data, which necessitates separating model management from data management. Ensuring data integrity requires constant synchronization between these systems, which in turn introduces consistency challenges. Document databases such as MongoDB necessitate converting YANG data into the BSON format, which is a memory-intensive process and involves frequent data format conversions^[5]. These conversions can degrade performance, especially in resource-constrained settings like network base stations.

In such environments, devices produce vast amounts of configuration and state data, intricately detailed by YANG models. Typically, this data is stored directly in relational database management systems (RDBMSs). While RDBMSs are built for handling structured data, they face numerous inefficiencies when dealing with YANG-modeled data structures. Most RDBMSs lack native capabilities for managing YANG models and validating documents according to YANG schemas. Consequently, developers are compelled to devise supplementary modules to facilitate model governance and data validation, introducing an additional layer of complexity to system architectures and increasing maintenance overhead. This necessitates a reevaluation of conventional data storage strategies to accommodate the intricacies of YANG models more effectively and streamline system design for enhanced scalability and maintainability.

Moreover, most network devices cannot afford high levels of memory and computing resource consumption. Given the limited performance of these devices, they may struggle to handle complex data operations, run advanced algorithms, or manage a large number of network connections simultaneously. In this resource-constrained scenario, the performance bottleneck becomes particularly evident when dealing with massive network data or executing compute-intensive tasks, which could lead to degraded performance or even system crashes.

These problems motivate us to design and implement a data-

base that efficiently stores and indexes YANG model data and can be deployed across diverse environments without imposing substantial resource demands. We build DB4YANG on an efficient key-value storage structure with a high-performance indexing mechanism for parsing, storing, and managing YANG-modeled data, thereby simplifying both data management and retrieval. The data management module specializes in handling data objects constrained by YANG models, validating the correctness of XML or JSON documents, and converting them into corresponding objects for storage. Besides, to maintain data integrity and consistency, DB4YANG adopts optimistic locking to achieve concurrency control. The development of an embedded database for YANG models in low-storage-performance network transmission contexts is of considerable practical and industrial importance.

2 Related Work

2.1 Yang Model Application

YANG has emerged as the industry-preferred schema for data description, prompting standards bodies, vendors, operators, and over-the-top (OTT) service providers to define their respective YANG models. This trend has garnered attention from prestigious universities and research institutions worldwide, which are actively engaged in developing applications based on YANG models and researching ways to enhance network transmission performance through YANG-driven methodologies.

Golmohammadi^[6] developed Trace, a centralized tracing tool that triggers traceroute commands with a distinct syntax on transmission control protocol/Internet protocol (TCP/IP) routers from a network management system. It tracked network hops and BGP autonomous systems (ASes), measured round-trip time (RTT) to specific endpoints, and sent the data back to the management system. The authors also evaluated the viability of integrating this tracing tool with YANG-based network management systems.

Lee et al.^[7] improved the NETCONF protocol using YANG XML documents. They devised a method for creating data blocks based on device configuration dependencies, allowing for the logical organization of configuration data. This data can be stored and managed in separate segments, enhancing efficiency when multiple administrators work on different data blocks simultaneously, which makes this enhanced NETCONF outperform existing methods.

Dallaglio et al.^[8] proposed a control and management plane for elastic optical networks (EONs) using NETCONF and YANG. They introduced a YANG model for optoelectronic transponders supporting sliceability, variable bit rates, and monitoring. Their experimental validation confirmed the model's effectiveness in controlling and managing EONs.

Amlou et al.^[9] created a microservice-based architecture for network automation, utilizing OpenConfig YANG models and NETCONF. This architecture simplified network configura-

tion, automated tasks, and reduced errors. Their work demonstrated how this approach streamlined configuration and data collection, effectively managing complex networks. Performance assessments confirmed its robustness in handling detailed configurations and its fault tolerance.

Liu et al.^[10] introduced MAYA, a system that aligns YANG models from various vendors or organizations by examining node attributes. MAYA uses semantic similarity techniques to measure the distance between attributes within nodes from different YANG models. They developed a YANG model alignment algorithm based on the stable marriage problem (SMP) to create inter-model mappings. Case studies and experiments showed that MAYA meets the alignment needs of YANG models in real-world settings.

In NETCONF- and YANG-based implementations of the data plane, validation efforts typically focus on YANG model verification. However, gaps in validation can lead to configuration errors, such as incorrect IP addresses or ports. Martinovic et al.^[11] proposed enhancements to data model validation by comparing actual configurations against expected ones, using XML comparison techniques.

Overall, current research on YANG models is largely centered on advancing YANG-based network management technologies and refining YANG's implementation and performance within modern network architectures^[12]. Despite this, there are still significant gaps in the direct governance and management of YANG model data.

2.2 Embedded System

In embedded systems, database operations are typically subject to strict timing constraints. Within this ecosystem, a class of database solutions exists that can be integrated into applications as library functions. These are known as embedded database management systems (EDBMS) or more commonly as embedded databases^[13]. By eliminating the overhead of client-server architectures, these systems offer efficient storage directly within applications and often act as local storage solutions. Noted for their minimal resource consumption, embedded databases are dedicated solely to their host processes and are capable of storing data volumes up to the terabyte scale. They are highly scalable, customizable, and strongly compatible across various platforms. Moreover, they are designed to be resilient in the face of failures.

Since embedded databases function on the application side without a dedicated maintenance team, they are vulnerable to user errors and hardware malfunctions^[14]. This necessitates the implementation of robust fault-handling strategies. Well-known examples of embedded databases include SQLite^[15], LevelDB^[16], and RocksDB^[17].

The primary difference between embedded and non-embedded databases lies in their operational models. Unlike standalone database servers that operate as separate daemons, embedded databases share the same process space as the ap-

plication. It is important to note that simply having a database on an embedded device does not make it an embedded database; such databases are more accurately described as embedded mobile databases. While it is theoretically possible for embedded devices to run server-side programs for network databases, the term "embedded databases" refers specifically to databases^[18] that run locally, integrated with the application without the need for a separate server instance, and are bound to the application's life cycle.

In contrast, conventional database access models employ intermediaries such as JDBC or ODBC^[19], through which clients communicate with remote database servers responsible for actual data manipulations. As depicted in Fig. 1, this model adheres to the classical client-server architecture, where clients and servers, as discrete processes, can collaborate closely whether they reside on the same machine or span different computers and networks.

Sysrepo is an open-source application that supports the required abstraction^[20]. To the best of our knowledge, Sysrepo is the most widely used database for managing and storing YANG-related data. It maintains a datastore, a core component of a NETCONF-enabled device. This datastore contains the configuration state of the device as defined by YANG data models installed within it. The code was modified to function as a plugin-style application for Sysrepo. This modification introduced a Sysrepo Client Library within the application, enabling it to subscribe to YANG modules in Sysrepo's datastore, with protocol buffers serving as the interface between the client library and the datastore engine. When a new NETCONF configuration utilizing a subscribed YANG data model is received, this setup triggers the execution of the C application.

Netopeer2, the third software system working in tandem with Sysrepo, functions as a NETCONF server^[21]. Operating as a background service (or daemon) on the switch, it monitors the management network for incoming NETCONF connections. Remote devices configure the switch via NETCONF by interacting with the Netopeer2 service. Upon receiving a NETCONF configuration update from a remote device, Netopeer2 processes the data payload and validates it against the

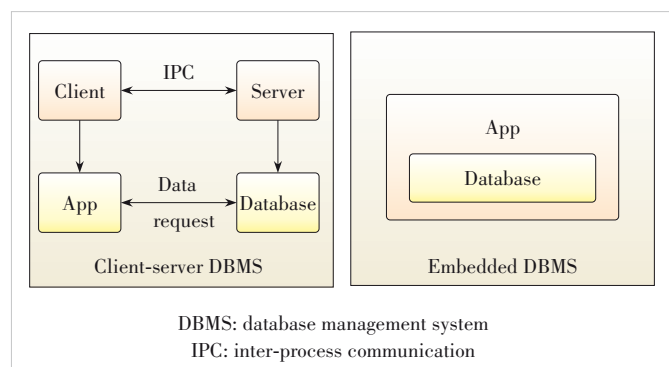


Figure 1. Architecture of client-server database and embedded database

YANG modules in the datastore to ensure consistency. After successful validation, the updated configuration settings are applied by invoking the relevant applications subscribed to the respective YANG model through Sysrepo.

3 Proposed Embedded Database Overview

3.1 Database Structure

The YANG model delegates functions to discrete modules, featuring an embedded design with minimal coupling. The architecture of the YANG model database facilitates client interaction through NETCONF and SNMP interfaces supported by the runtime environment. These interfaces convey YANG models and data to a database service centered around a data storage module. This service is meticulously designed to administer both model management and data management tasks.

The model management module, a core component, directly oversees the handling of YANG model files. Its responsibilities encompass the import/export, deletion, and modification of models, thereby ensuring the integrity and flexibility of the schema definitions. The data management subsystem is tasked with the storage of documents conforming to YANG model constraints within the database and adopts an optimized schema. It further implements specific routines for constraint validation and for performing create, read, update, and delete (CRUD) operations at the node level, enhancing data integrity and operability.

Presently, the model management and data management modules operate independently. These modules perform periodic synchronization of their datasets to maintain coherence. To augment functionality, an additional plugin subsystem that endows the system with data subscription capabilities has been integrated into the framework. Leveraging this subscription mechanism, affiliated business services can promptly react to modifications in the model data, thereby enabling real-time responsiveness.

Fig. 2 shows that the architecture of DB4YANG is constructed with different modules, each dedicated to a distinct function. It comprises the storage service module, the data management module, and the model management module, all of which rely on the B+ tree. Collectively, these modules form the backbone that primarily facilitates advanced model governance, robust data handling, and dynamic data subscription functionalities. Thereby the proposed database is a sophisticated and adaptable database infrastructure tailored for YANG-based network configurations.

3.2 Schema Module

DB4YANG employs a stratified approach to managing data. This system abstracts the minimal ensemble of YANG model files capable of describing a category of network devices into a “Container” for that category of devices. As YANG model files undergo parsing and type transformation, this Container

accumulates a multitude of “Types”, which may originate from typedef, container, or grouping nodes within the YANG schema. Guided by the dependencies among these types, the system constructs an intricate type hierarchy that serves as the basis for device description.

Acting as an abstraction of devices, a Container can be instantiated as a “Device”, representing a concrete, real-world network device comprising its entire informational footprint. Similarly, XML/JSON documents can be parsed and transformed into objects of a corresponding type, reflecting the concept of type instantiation. However, not all types possess the capability to interpret such documents. This capability is typically attributed to types derived from nodes such as container and RPC, highlighting their distinctive role in data interpretation and object creation.

The process of schema modification requires a base data container B and a schema modification sequence Q as input, where Q is a series of batch schema modification operations. Typical operations include inserting a new YANG file and removing constraints from a YANG file.

The schema modification process is divided into three phases: preparation, execution, and cleanup.

1) Preparation phase: Check if there is an ongoing schema modification on the base data container B ; if so, halt the process. Otherwise, create a backup of B 's model, retrieve all YANG files from B , and apply the schema modification sequence to generate a new set S of YANG files. Finally, create an execution thread.

2) Execution phase: Create a new data container C based on B 's data store, and attempt to add the YANG files from set S to C sequentially. If all additions succeed, the execution is

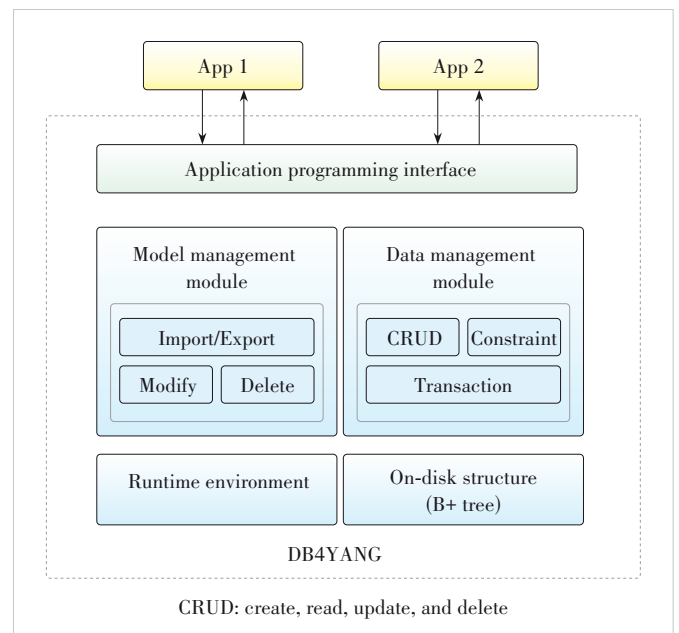


Figure 2. DB4YANG architecture

considered successful; if any addition fails, delete C .

3) Cleanup phase: If the execution succeeds, replace B with C , and update the metadata and YANG file set in external storage. If the execution fails, clean up the newly generated set S of YANG files.

3.3 Data Module

Compared with other databases, DB4YANG is an XML/JSON document database that operates under YANG schema constraints. It is required that specific data management operations be executed only on data containers where a schema tree definition has been established. In other words, any data management operation performed on a data container that has not undergone a model insertion operation will not succeed.

DB4YANG encodes documents that have passed validation into DB4YANG tuples and stores them in a B+ tree. The B+ tree is an N-ary sorted tree in external memory, with insertion and modification operations exhibiting relatively stable logarithmic time complexity. A typical B+ tree provides common key-value query interfaces such as GET, PUT, and RANGE. To adapt to the form of the B+ tree interface, DB4YANG assigns a unique identifier as a key to each piece of data before insertion, with the DB4YANG tuple serving as the value. The logical structure of the DB4YANG tuple is as follows: TXID, CTR_VER, DFLAG, and VER_PTR are all fixed-length 8-byte integers (stored in little-endian format). These fields are collectively used for the implementation of a multi-version concurrency control mechanism and document re-validation; DATA represents the actual stored document, encoded in the length and value (LV) format, with the length field encoded as a fixed-length 4-byte integer.

Furthermore, we propose a high-performance indexing design scheme that draws on the principles of inverted indexing (as illustrated in Fig. 4). This paper constructs a reverse mapping structure that maps query paths (QPATH) to object identifiers (OIDs), specifically HANDLE (analogous to document IDs in inverted indexes), thereby effectively simulating the fast query logic of inverted indexes. In particular, this paper introduces bitmap technology to further optimize HANDLE storage, which improves space efficiency.

Serving as an abstraction layer between users/applications and the underlying data storage, the external schema provides a concise and intuitive interface for data access. It is represented by a correspondence between a QPATH and an INDEX_PATH. Specifically, the QPATH is an XPath expression used to locate specific YANG model elements, where XPath serves as the standard notation for expressing query paths. The INDEX_PATH represents an internal identifier that points to a dedicated B+ tree. For each INDEX_PATH, a set of OIDs is persistently stored in a B+ tree

structure, with these OIDs corresponding to all documents matched by the XPath query.

From the perspective of the data management module, regardless of how complex the internal structure of the node-level index may be, its external representation remains a mapping of XPath to the B+ tree storing OIDs. This design allows external users or systems to access and manipulate indexes in a unified and straightforward manner, without concerning themselves with the underlying implementation details.

As shown in Fig. 5, DB4YANG adopts a refined and efficient metadata structure management strategy, centered on the use of a hash table to organize and access index metadata. The metadata is further stored in a B+ tree structure whose file path is /tmp/indexes/01.bp. This B+ tree not only ensures efficient data retrieval speeds but also carries key information about the index structures corresponding to each QPATH. Currently, the critical information in each metadata record is the actual storage location of the index (INDEX_PATH), which indicates the centralized storage point for all related HANDLES (existing in an OID form). To efficiently utilize storage space and accelerate query processing, this paper introduces an innovative interval partitioning scheme.

Considering that a HANDLE (OID) is a 64-bit integer,

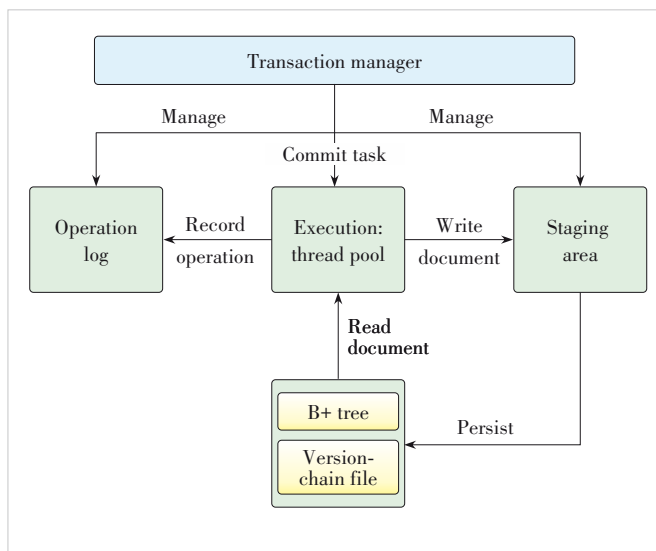


Figure 3. Structure of data management module

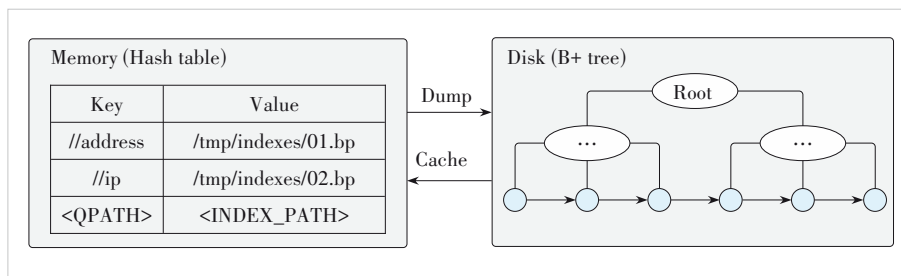


Figure 4. External view of index storage

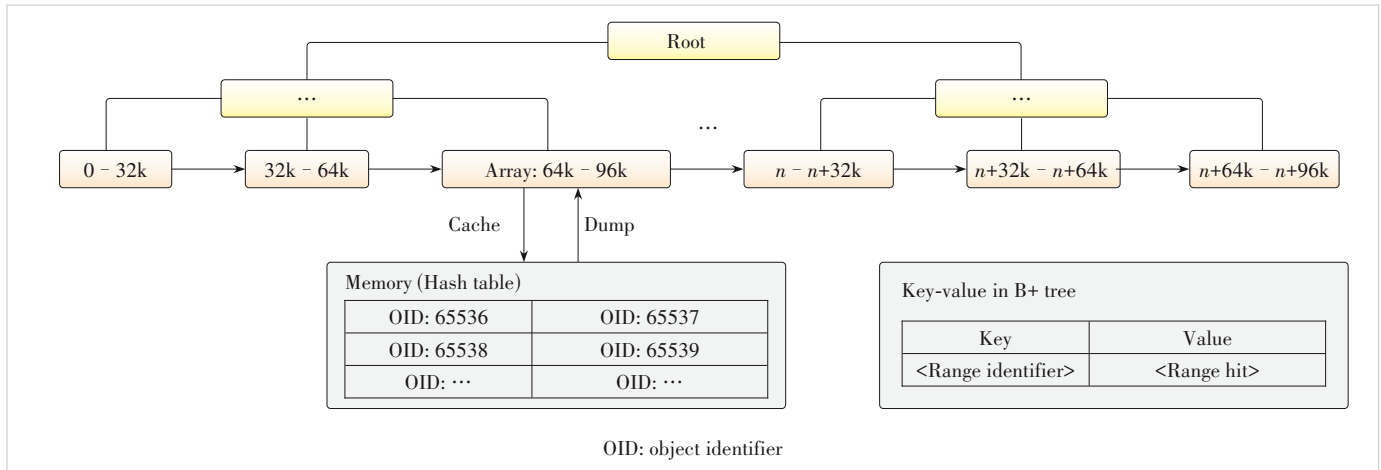


Figure 5. Internal view of index storage

this paper divides its numerical range into 2^{49} fixed-length subintervals, each of which corresponds to a fixed-size page of 4 096 bytes (i. e., one page), and uses the smallest HANDLE value within the interval as the unique interval identifier. This partitioning method not only simplifies the logic of index lookup but also improves the efficiency of memory and disk I/O operations through the preset fixed-size pages.

To store a HANDLE with a specific value of h , the system first determines the interval to which h belongs. Then, using bitmap technology, the system encodes and stores the set of HANDLES within the interval, optimizing storage space usage and enabling fast bitwise operation queries. In the bitset, each bit corresponds to a HANDLE, indicating its presence or absence, which greatly compresses the data volume. It also allows HANDLES to be inserted, deleted, and queried directly through bitwise operations, thereby enhancing processing speed.

4 Transaction Management

The YANG modeling language is designed for network device configuration, and most of its data definitions and operations are aligned with network management protocols such as NETCONF and RESTCONF. These include configurations, state data, remote procedure calls, and notifications. For network configurators, the process of setting up a device typically involves obtaining the corresponding YANG module and issuing configurations according to its schema. By contrast, developers might be tasked with authoring YANG modules, which requires adherence to YANG's syntax rules. Such tasks often necessitate meticulous drafting and verification by network operations personnel to ensure a high degree of operational accuracy in real-world deployment and application, thereby maintaining operational efficiency.

In practice, during the configuration of network devices, environments with high concurrency and intense data contention

are rarely encountered. Common application scenarios involve an individual network administrator batch-updating multiple network devices with configuration information or subscribing to device alerts. These operations typically involve minimal data conflicts, incur low retry costs, and require no absolute data consistency. Under such working conditions, optimistic locking is a more suitable choice than pessimistic locking for implementing concurrency control and transaction management.

A transaction is the unit of concurrency control in DB4YANG, defined as a sequence of operations issued by an application, and its execution satisfies the atomicity, consistency, isolation, and durability (ACID) properties. Fig. 6 shows that the DB4YANG data management module consists of components including a transaction manager, thread pool, staging area, operation log, B+ tree, and commit log, with the transaction manager at its core. DB4YANG provides a global default transaction manager while also allowing applications to use custom transaction managers. The set of transactions managed by a transaction manager is called the transaction group of that manager, and different transaction groups should be isolated from one another to guarantee inter-group transaction isolation. Concurrent transactions within a transaction group share the same thread pool, operation log, B+ tree, and commit log, while each transaction maintains its own staging area to stage the document modifications made by that transaction.

The operation log records significant actions during transaction execution processes, including the initiation, rollback, and commit of transactions, as well as the insertion, update, and deletion of documents. It is utilized for fault recovery upon a crash and subsequent reboot of DB4YANG. Based on the transaction execution characteristics of DB4YANG, the operation log is designed as a mixed UNDO/REDO log. Each operation log entry is encoded in binary for storage and comprises six fields: the transaction ID, operation type, container name, OID, old value, and new value.

When a transaction is selected by the scheduler, it transi-

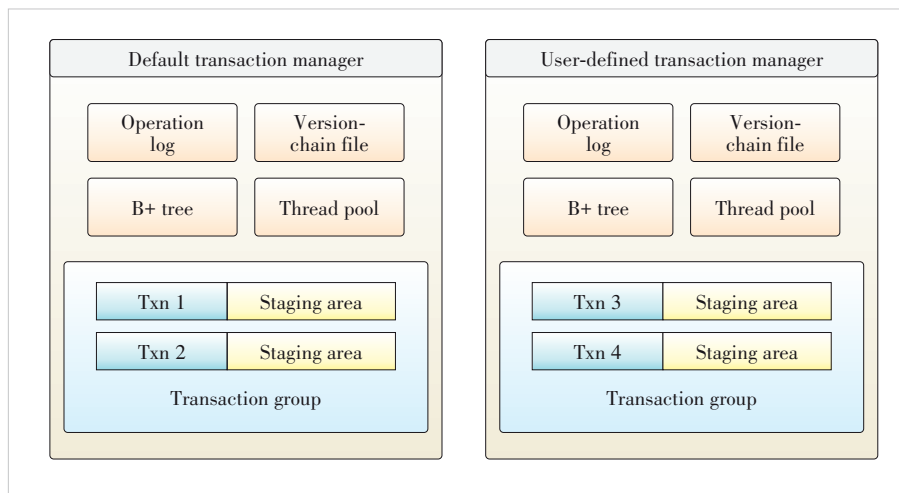


Figure 6. Transaction manager and transaction group

tions from the ready state to the running state. During this period, the transaction begins executing its database operations, such as queries, inserts, updates, or deletions. The system ensures the atomicity of operations within a single transaction, meaning that all operations either succeed entirely or fail altogether.

Once all operations within a transaction have been successfully executed, the transaction enters the pre-commit phase, a critical checkpoint for safety. Here, the transaction manager verifies the results of the transaction's execution and the current state of the system, ensuring that the transaction meets all predefined conditions and constraints (such as foreign key constraints and uniqueness constraints). This step is crucial for maintaining the consistency of the database.

If the pre-commit check succeeds, the transaction is formally committed. The system updates the commit log to record the transaction's success, a step that is key to achieving durability. Following this, the database modifications associated with the transaction have been permanently persisted, which marks the end of the transaction.

In contrast, if the commit check fails due to business rule violations or the detection of potential conflicts, the transaction is aborted. The system must then perform a rollback operation to revert the database state to its condition prior to the start of the transaction, ensuring that data consistency remains intact. In this process, all executed operations are recorded in the operation log, providing the necessary information for any subsequent retry or recovery.

In the event of unexpected exceptions during the transaction execution phase, such as system errors or insufficient resources, the system immediately triggers the rollback process of the transaction. All completed operations are rolled back sequentially, and the redo log and undo log (if applicable) are updated accordingly to record these changes and ensure the atomicity and durability characteristics of the transaction.

The data affected by the transaction is thereby restored to its initial state, as if the transaction had never been executed.

This transaction processing flow demonstrates how, through stage delineation and state management, data reliability and transaction-handling efficiency can be ensured amid complex and dynamic database operations.

5 Experiment

Sysrepo is a YANG-based configuration and operational state data store for Linux applications. Applications can use Sysrepo to store their configuration modeled by a provided YANG model instead of relying on flat configuration files, for example. Sysrepo ensures the consistency of the data stored in the

datastore and enforces the data constraints defined by the YANG model. Sysrepo can be easily integrated with management agents such as NETCONF or RESTCONF servers, using the same client library API that applications use to access their configuration. To our knowledge, Sysrepo is widely used to manage and store YANG-related data.

In this section, we verify system performance by comparing each operation with its counterpart in Sysrepo. We evaluate the schema modification, document insertion, and document reading performance of DB4YANG through automated programs, analyze the performance issues present in the current version, and discuss relevant improvement strategies.

To evaluate the performance of our database and Sysrepo, we conducted comparative experiments on a workstation equipped with an AMD Ryzen 7 4800H processor (8 cores, 4.2 GHz), 16 GB of DDR4 DRAM, and 512 GB of SSD storage.

In Figs. 7 and 8, the performance of the proposed database significantly surpasses that of Sysrepo for the following reasons. In Sysrepo, the verification of content changes is implemented through string comparison (see the function `lyd_anydata_equal`). When the configuration is large, the generated strings become lengthy, making comparisons time-consuming.

Configurations of subordinate modules that share the same top-level module are not segregated into separate files; instead, they are stored together in the top-level module's file. Consequently, when the data volume of one subordinate module grows large, loading this shared file becomes slow, which also degrades the performance of other modules.

Every modification and save operation involves rewriting the file. I/O operations are relatively slow, and the larger the data volume, the longer the time required. In addition, the shared memory file lock means that as long as one process is reading a large configuration, the lock is held, stalling other processes. YANG data is structured as a tree, in which the children of

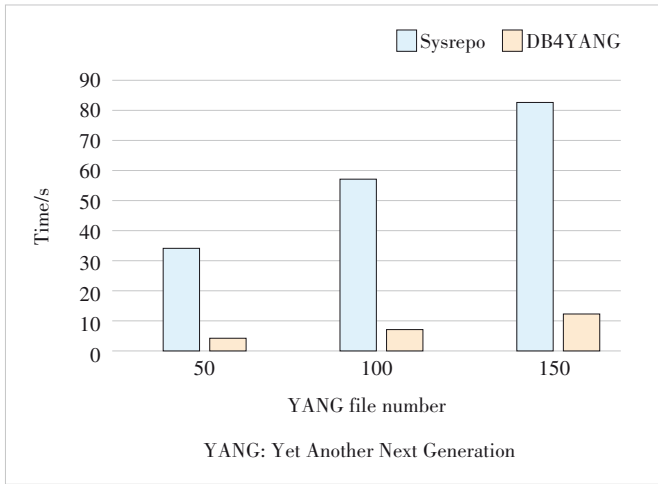


Figure 7. YANG schema import efficiency comparison

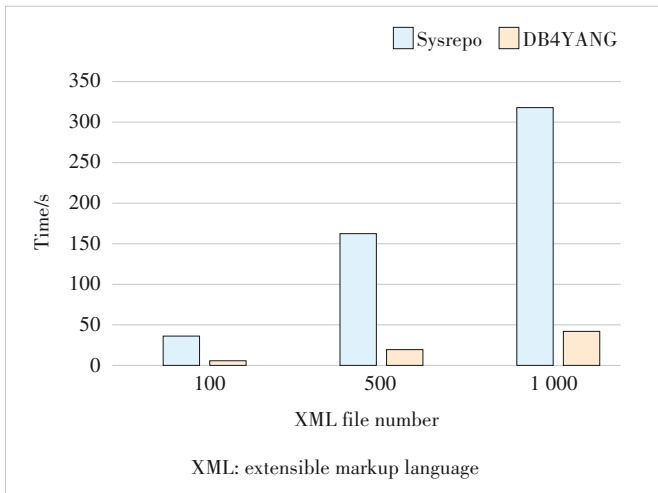


Figure 8. YANG document writing efficiency comparison

each node are organized as a doubly linked list. Since traversing a linked list requires linear time, this organization increases the latency of data retrieval and modification.

To optimize the performance of DB4YANG during system configuration, we load data for the requested module and its dependent modules for validation based on the configuration request. Due to interdependencies among configurations, any changes may impact related configurations, which must be considered during operations.

In the configuration-handling logic, changes must be synchronized across all submodules when operations such as deleting or replacing a top-level module's configuration are performed. This necessitates loading the data of all modules under the top-level module to ensure the correctness and consistency of the configuration. This step is essential to guarantee the accuracy of the configuration.

DB4YANG's caching functionality has also been opti-

mized. Previously, the entire data forest of the running datastore was cached as a single copy; in the optimized version, the cache is modularized. For the top-level modules of the project, we only cache their own data. For other top-level modules and their sub-modules, which are accessed less frequently, the modular splitting approach is not employed, and their data is cached as a whole. In this way, the modular caching strategy ensures efficient data management while reducing the complexity of cache handling.

To verify the transactional capabilities of our proposed database, we conducted an experiment, the results of which are depicted in Fig. 9. When the number of transactions reached 1 500, we executed a series of operations 100 times, comprising three transactional insertions, one point query, one full query, four XPath document-level queries, and four XPath node-level queries. Additionally, we performed one point-update operation and one point-deletion operation, each followed by a verification query to confirm that the update or deletion had taken effect. In total, these 15 operations constituted a single transaction. Following this pattern, when the number of transactions reached 15 000, the same set of operations was repeated 1 000 times; when it reached 150 000, the operations were executed 10 000 times.

Fig. 9 shows the influence of transaction numbers. Repeated experiments revealed that when all operations were XPath document-level queries, the transaction execution speed dropped to half of the average. Furthermore, in the absence of data deletion, queries gradually slowed down as data accumulated. As the data volume grows, querying increasingly becomes the critical factor limiting performance.

The main thread completes the model loading process, which does not become a bottleneck for the system. Concurrent read transactions are handled by child threads. And the functions `lyd_parse_data_mem` and `lyd_free_all` constitute the main bottleneck of the system.

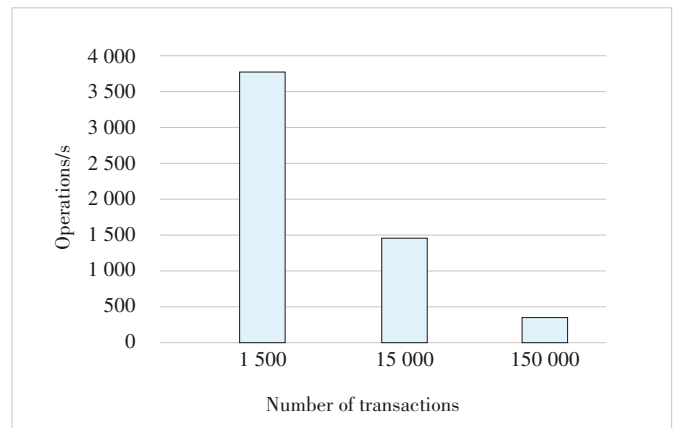


Figure 9. Performance of complicated transactional operations

Perf¹ is a performance analysis tool for Linux that can perform function-level and instruction-level hotspot detection. It was used to analyze the CPU usage of hotspot functions in programs, thereby locating performance bottlenecks.

Fig. 10 shows that in DB4YANG, the program spent 60.48% of its time loading YANG schemas and 22.82% on document insertion and reading. However, in Sysrepo, the majority of the program's runtime was consumed by loading YANG schemas. Due to the relatively small data volume in this test (only 100 documents), model loading time accounted for a larger proportion of the total time. In real-world scenarios, YANG models are only loaded after a restart, which will not cause a performance issue.

In DB4YANG, document parsing and data tree serialization have become performance bottlenecks. During document insertion and reading, the functions `lyd_parse_data_mem` and `lyd_print_mem` occupied 18.19% and 3.01% of the program's runtime, respectively. Together, they accounted for approximately 93% of the time spent in the document insertion and reading phase, which itself took up 22.82% of the total runtime. The function `lyd_parse_data_mem` is used to load document data of a certain format (e.g., LYB, JSON, or XML) into a data tree in memory, while `lyd_print_mem` converts the data tree into document data (e.g., JSON or XML). For simplicity, these two functions are referred to as the document parsing function and the data tree serialization function, respectively, in the following discussion.

Analysis of Fig. 10 reveals that the Sysrepo functions `sr_modinfo_data_load`, `lyd_validate_modules`, and `sr_modinfo_data_store` are primarily responsible for the increased execution time. In large-scale configuration scenarios, the volume of saved configuration data significantly increases. Consequently, this leads to longer processing time for data loading, module validation, and data storage.

6 Conclusions

This paper focuses on resource-limited settings such as base stations and explores the implementation of an embedded database system for managing YANG model data. We have designed and implemented high-speed read and write operations tailored for YANG model management and its associated data. We propose optimization methods to mitigate performance degradation caused by using relational databases for YANG model data, which involves non-native processing and format conversion. This directly addresses the technical hurdles of traditional methods, such as storing YANG model data in relational tables, which requires conversion between 2D relational data and YANG model data, incurring unnecessary overhead.

Furthermore, we present strategies for creating a lightweight, high-performance indexing system. This system not

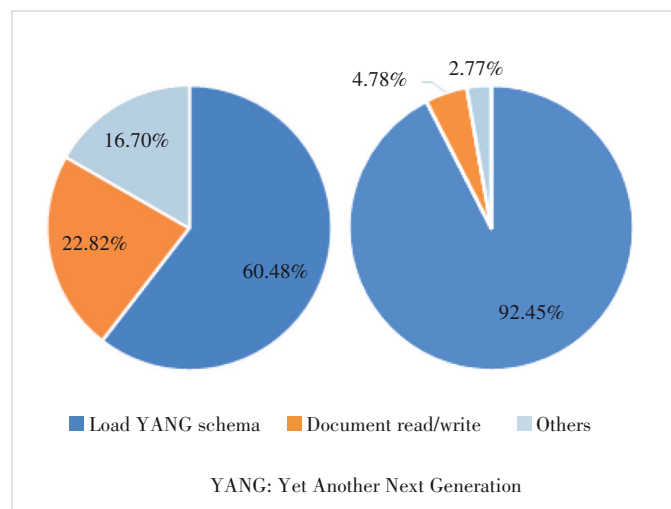


Figure 10. Perf test of DB4YANG and Sysrepo

only boosts data access speeds but also excels in space efficiency, making it ideal for handling extensive datasets. Through effective index structures, we have achieved swift data retrieval and enhanced query performance while keeping storage usage in check, making it suitable for environments with limited resources.

Additionally, this paper delves into transaction processing for YANG model data, adhering strictly to ACID principles to ensure data integrity—a critical factor in databases facing high concurrency and complex operations. Our data management module operates on a transactional basis. Data is valid upon transaction commit and void if the transaction is rolled back. We have developed a transaction management system for DB4YANG to fulfill ACID requirements.

Experimental results indicate that DB4YANG outperforms existing methods in terms of accuracy and efficiency. Moreover, its transaction management meets industry standards and demonstrates its viability for real-world applications. However, our research has certain limitations. For instance, performance is somewhat constrained by the parsing speed of YANG model documents using the `libyang` library. This can lead to bottlenecks during document parsing and conversion. In future work, we aim to enhance the database's performance and scalability by rewriting the YANG model's parsing and processing functions.

References

- [1] Yi X X, Zhang N H, Liu Y C, et al. Key technologies for intelligent computing power network [J]. ZTE technology journal, 2025, 31(2): 31–38. DOI: 10.12142/ZTETJ.202502005
- [2] Zhu L H, Karim M M, Sharif K, et al. SDN controllers: a comprehensive

¹ www.brendangregg.com/linuxperf.html

- analysis and performance evaluation study [J]. *ACM computing surveys*, 2021, 53(6): 1 – 40. DOI: 10.1145/3421764
- [3] Wang H. Improvement and implementation of wireless network topology system based on SNMP protocol for router equipment [J]. *Computer communications*, 2020, 151: 10 – 18. DOI: 10.1016/j.comcom.2019.12.038
- [4] Bhat M G, Bhattacharjee S, Gündoğan C, et al. CORECONF, NETCONF, and RESTCONF: benchmarking network orchestration in constrained IIoT devices [J]. *IEEE Internet of Things journal*, 2024, 11(7): 13082 – 13090. DOI: 10.1109/jiot.2023.3338470
- [5] Györfi C A, Dumşeu-Burescu D V, Zmaranda D R, et al. A comparative study of MongoDB and document-based MySQL for big data application data management [J]. *Big data and cognitive computing*, 2022, 6(2): 49. DOI: 10.3390/bdcc6020049
- [6] Golmohammadi Z. Centralized model driven trace route mechanism for TCP/IP routers: remote traceroute invocation using NETCONF API and YANG data model [D]. Tampere: Tampere University, 2019
- [7] Lee Y M, Lee J K. YANG-based data modeling techniques for the content layer of NETCONF to improve query throughput [J]. *Journal of Internet technology*, 2018, 19(7): 1993 – 2007. DOI: 10.3966/160792642018121907004
- [8] Dallaglio M, Sambo N, Cugini F, et al. Control and management of transponders with NETCONF and YANG [J]. *Journal of optical communications and networking*, 2017, 9(3): B43. DOI: 10.1364/jocn.9.000b43
- [9] Amlou A, Abane A, Merzouki M, et al. Automated network programmability using OpenConfig YANG models and NETCONF protocol [C]//The 20th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA). IEEE, 2023: 1 – 5. DOI: 10.1109/aiccsa59173.2023.10479244
- [10] Liu Y B, Dong Y Q, Shen J, et al. MAYA: exploring multi-form attributes of node to align YANG data models [J]. *Computer networks*, 2023, 222: 109538. DOI: 10.1016/j.comnet.2022.109538
- [11] Martinovic B, Vuleta M, Popic S, et al. Simple verification improvements of the NETCONF data model in programmable computer Networks [C]//The 30th Telecommunications Forum (TELFOR). IEEE, 2022: 1 – 4. DOI: 10.1109/telfor56187.2022.9983688
- [12] Guliato D, de Melo E V, Rangayyan R M, et al. POSTGRESQL-IE: an image-handling extension for PostgreSQL [J]. *Journal of digital imaging*, 2009, 22(2): 149 – 165. DOI: 10.1007/s10278-007-9097-5
- [13] Marwedel P. Embedded system design: embedded systems foundations of cyber-physical systems, and the Internet of Things [M]. Cham: Springer International Publishing, 2021. DOI: 10.1007/978-3-030-60910-8
- [14] Narayanan V, Xie Y. Reliability concerns in embedded system designs [J]. *Computer*, 2006, 39(1): 118 – 120. DOI: 10.1109/mc.2006.31
- [15] Gaffney K P, Prammer M, Brasfield L, et al. SQLite: past, present, and future [J]. *Proceedings of the VLDB endowment*, 2022, 15(12): 3535 – 3547. DOI: 10.14778/3554821.3554842
- [16] Wang L, Ding G Q, Zhao Y L, et al. Optimization of LevelDB by separating key and value [C]//The 18th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT). IEEE, 2017: 421 – 428. DOI: 10.1109/pdcat.2017.00074
- [17] Dong S Y, Kryczka A, Jin Y Q, et al. RocksDB: evolution of development priorities in a key-value store serving large-scale applications [J]. *ACM transactions on storage*, 2021, 17(4): 1 – 32. DOI: 10.1145/3483840
- [18] Chauhan A. A review on various aspects of mongodb databases [J]. *International journal of engineering research & technology (IJERT)*, 2019, 8 (5): 90 – 92
- [19] Bai Y. JDBC API and JDBC drivers [M]//Oracle database programming with Java: ideas, designs, and implementations. Boca Raton: Auerbach Publications, 2022: 67 – 86. DOI: 10.1201/9781003304029-3
- [20] Sysrepo Project. Sysrepo.org [EB/OL]. [2024-02-14]. <http://www.sysrepo.org>
- [21] Ahearne S, Verbishchuk Y, Sreenan C, et al. Software defined control of tunable optical transceivers using NETCONF and YANG [C]// European Conference on Networks and Communications (EuCNC). IEEE, 2018: 1 – 86. DOI: 10.1109/eucnc.2018.8443188

Biographies

Han Bing is currently pursuing his MS degree in electronic information at the National Elite Engineers School, Harbin Institute of Technology, China. He specializes in research directions such as database system design and edge storage.

Gao Jirun is currently pursuing his MS degree in the Department of Computing, Harbin Institute of Technology, China. He focuses on research areas such as database systems and edge storage.

Zhang Kaiqi (zhangkaiqi@hit.edu.cn) is a lecturer at the Department of Computing, Harbin Institute of Technology, China. His research interests include spatiotemporal big data query, approximate computation, and embedded database systems. He has published 15 papers.

Wang Hongzhi is a professor at the Department of Computing, Harbin Institute of Technology, China. His research interests include big data governance, big data management and analysis, database systems, and industrial big data. He has published more than 300 papers.

Chen Hedui is a senior engineer at ZTE Corporation. He specializes in databases, new hardware acceleration, lakehouse integration, and AI for databases.