The Next Generation Operational Data Historian for IoT Based on Informix

Sheng Huang, Yaoliang Chen, Xiaoyan Chen, Kai Liu, Xiaomin Xu, Chen Wang
IBM China Research Lab
{huangssh, yaoliangc, xiaoyanc, LiuKaiIBM, xuxxm, wangcwc} @cn.ibm.com

Kevin Brown, Inge Halilovic
IBM Informix Software Team
{kbrown3, ingeh}@us.ibm.com

ABSTRACT

In the era of the Internet of Things (IoT), increasing numbers of applications face the challenge of using current data management systems to manage the massive volume of operational data generated by sensors and devices. Databases based on time series data model, like PI Server, are developed to handle such data with operational technology (OT) characteristics (high volume, long lifecycle, and simple format). However, while achieving excellent write performance, these database systems provide limited query capabilities. In this paper, we present the next-generation Operational Data Historian (ODH) system that is based on the IBM© Informix© system architecture. The system combines the write advantages of the existing time series databases and the ability to run complex queries in SQL. We demonstrate the high efficiency of our system for both writing and querying data with a variety of case studies in the industries of Energy and Utilities and connected vehicles. In addition, we present the first benchmark, IoT-X, to evaluate technologies on operational data management for IoT.

Categories and Subject Descriptors

H.2.4 [**Information Systems**]: Database Management – *Systems*

Keywords

Relational DB, Time series DB

1. INTRODUCTION

According to a Gartner prediction, by 2015 more than 70% of CIOs will need to oversee the performance of connected digital technologies. Enterprise services increasingly demand accurate, coherent, and timely information. We will witness the hyper-growth of instrumented devices for sensing the world, along with the requirement for infrastructure to connect and manage the devices and the sensor data that the devices collect. Intelligence will be added in the field and in the data center. Significant growth opportunities will arise in end-to-end infrastructure and services. IDC forecasts that 15 billion devices will be communicating over the network by the year 2015. Devices are spreading like a digital nervous system as they become smaller, smarter, and cheaper. Big data grows increasingly bigger: All these "things" (sensors) will produce far more data than we have now, which will tax already complex enterprise information management systems.

As sensors become ubiquitous and inexpensive, the sampling data from sensors consume a large share of the storage space in big data management. There are many types of data formats in sensor-based applications in the heavy asset management industries, such as Chemical and Petroleum (C&P), Energy and Utility (E&U), connected vehicles, environment monitoring, etc. Huge volumes of data are generated in the format of scalar data or time series data, especially in C&P, E&U, and other plant management applications. There are two typical data characteristics scenarios in these applications. In one scenario, the number of sensors is not large, but the sensors have high sampling rates (>1Hz). A typical oil detection application, for example, can have thousands of sensors collecting data simultaneously with a sample rate of about 500Hz for each sensor. The other scenario, in contrast, involves a massive amount of sensors that have low sampling rates (<1Hz). A smart meter system in one province in China might possess 10-50 million sensors with a 15-minute sampling interval for each meter. Both scenarios need to store historical data for long periods, process real-time queries, and analyze historical data.

Traditional relational databases, like DB2, Oracle, SQL Server, Sybase, and so on, dominated the information technology (IT) data management market for decades because of their rich query capabilities. However, for a 10 million smart meter scenario, a traditional relational database requires a 48-core rack cluster to meet real-time write requirements [6]. A parallel DBMS (or multiple single-node DBMSs), such as HadoopDB [1], can write data faster, but has poor fault tolerance [2]. In contrast, time series databases (TSDB), such as TempoDB [11], and real-time historians, such as PI Server [5], are robust and able to handle the load of writing data using many fewer CPU core servers and less storage. For simplicity, we consider real-time historians as TSDBs because they adopt a time series data model. A TSDB system can be adequate for big operational data management. However, a typical OT application must analyze business data that is stored in relational databases together with the accumulated OT data. Thus, relational databases and TSDBs must be combined in OT applications, which lead to redundant investment in data management systems. Moreover, migrating data from TSDBs to relational databases for analysis involves a huge computing overhead.

Our next generation ODH system combines write performance and query capabilities. Although some of the related technologies have been studied in academia, to the best of our knowledge, we are the first to combine these two aspects to build an operational data historian with the following key features:

High write throughput. We designed a new storage component inside of Informix that adopts a novel data model that dramatically reduces the I/O and storage size. High write throughput of up to millions of data points per second is achieved for both high frequency sensors and massive amounts of low frequency sensors. Balanced read/write optimization on the data store enables fast query performance.

Compressed and transparent data store. After buffering, writing, and compressing, the data have a 10-100 compression ratio. Data are viewed through a virtual table that has a simple relational schema, while the internal data store structures and the data organization are transparent to users.

Fast query performance. High read performance is obtained on the typical types of queries on operational data: slice queries on data generated by multiple data sources for a short time window, and historical queries on data generated by a specified data source for a long time window.

Operational and relational data fusion. Both relational data and operational data are stored in one database. The unified data access interface of SQL supports data extraction and fusion from both operational data and relational data. Using one database also eliminates the cost of migrating an existing system from a relational database to TSDB when the relational database in the legacy system cannot meet the write throughput requirement.

The rest of this paper is organized as follows: Section 2 describes our data model that supports efficient storage and querying. Section 3 provides a brief description of the system architecture to fulfill the design. Section 4 describes customer cases. The first IoT benchmark is shown in section 5, while section 6 concludes.

2. OPERATIONAL DATA MODEL

The ODH system handles real-time time series operational data records that are characterized as having high volumes, long lifecycles, and simple formats. Such data are typically generated by a sensor or a device, which we call a *data source*. Each data source produces *operational records* with a fixed *data schema*. The data sources that share the same data schema are considered to have the same *schema type*. Each operational record of the data contains the following information: a timestamp that indicates the time point when the data source collected the record; an id that identifies the data source of the record; and a couple of attributes for the measurements collected by the data source. For example, a sensor that monitors the surrounding environment could produce operational records with a data schema of: (*timestamp*, *id*, *temperature*, *wind*).

The *relational data model*, as adopted by relational databases, stores each operational data record as a tuple in a relational table. This approach leads to redundant storage costs for both ids and timestamps. A typical relational database can process around several 10,000 of records per second. The *time series data model* organizes operational data records by data source. This model is adopted by most of the TSDBs. The *id* of the data source is only stored once and the *timestamp*s can be stored as the delta values to their previous values, which requires fewer bits. However, operational data, as represented in the time series data model, are difficult to analyze in combination with relational data. If the number of the data sources is very large, it is not trivial to join the id between the time series data model and the relational table for a given time or interval.

In our ODH system, we adopt a novel data model that combines the advantages of both of the two approaches. It operates on data in a way that is similar to the time series model and presents the data to users as a standard relational schema. It stores the operational data adaptively, according to the data characteristics, into three different structures: a Regular Time Series (RTS) batch structure, an Irregular Time Series (IRTS) batch structure, and a Mixed Grouping (MG) batch structure. Table 1 lists the relationships between the data characteristics and the batch structures that are used by the data model. A *regular* data source generates a time series with identical sampling intervals, while an *irregular* data sources generates a time series with variable sampling intervals.

Table 1. The batch structures vs. data sources and operations

Data Source	*Ingestion*	*Slice Query*	*Historical Query*
Regular high frequency	RTS	RTS	RTS
Irregular high frequency	IRTS	IRTS	IRTS
Regular low frequency	MG	MG	RTS
Iregular low frequency	MG	MG	IRTS

In operational applications, it is very common for a query to be interested in only a small number of tags out of a schema type that contains a few hundred tags (in TSDBs, tags are popularly used to represent attributes and granularity for pricing, where each value generated by a tag is regarded as a data point). Our operational data model adopts a tag-oriented approach to address this problem.

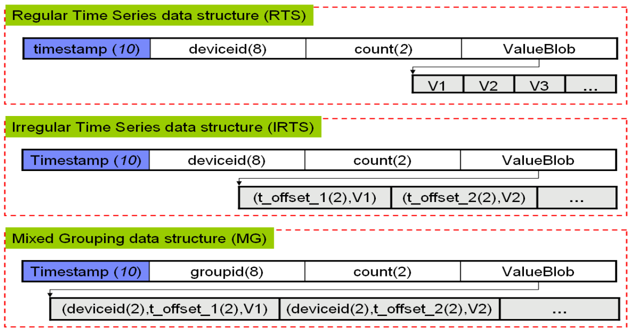

Regular Time Series data structure (RTS)
timestamp (10)
deviceid(8)
count(2)
ValueBlob
V1
V2
V3
...
Irregular Time Series data structure (IRTS)
Timestamp (10)
deviceid(8)
count(2)
ValueBlob
(t_offset_1(2),V1)
(t_offset_2(2),V2)
...
Mixed Grouping data structure (MG)
Timestamp (10)
groupid(8)
count(2)
ValueBlob
(deviceid(2),t_offset_1(2),V1)
(deviceid(2),t_offset_2(2),V2)
...

Figure 1. The Data Structures Used by the ODH Data Model

Figure 1 shows the three kinds of batch structures of the hybrid time series and relational data model. B-tree indices are created on the first two fields of each of the batch structures. The three batch structures reduce the I/O cost by reducing the number of records and, accordingly, the index size. The RTS and IRTS structures pack every *b* operational data points from one data source into a *ValueBlob* in a batch structure, where *b* is the batch size set by the user. In comparison, the MG structure packs *b* operational points by timestamp from a group of data sources into one record. All three structures minimize I/O operations by packing a number of operational points into one record. The structures also reduce the size of the data by using data grouping to compress ids and timestamps.

3. THE SYSTEM

An overview of ODH system architecture is shown in Figure 2. The system is based on Informix data servers. The three ODH components cooperate efficiently to successfully fulfill the requirements of high read and write efficiency and ease of use. The operational data model introduced in section 2 is used as the underlying storage model.

The ODH configuration component manages the metadata of the data sources and the Informix data servers that can be accessed by the storage and query components.

The ODH storage component ingests the operational data from devices and sensors through a set of carefully designed writer APIs that are highly efficient for the operational data model. The insertion process does not support transactions, but based on our engagement with real customers, transactions are usually not required in IoT data scenarios. If reasonable data loss is acceptable, the compressor component compresses the operational data before the data are stored. As described in section 2, the data model itself provides natural compression for the ids and the timestamps. However, the largest and most important portion of the operational data is the tags. It is not trivial to compress *tags* with high efficiency.

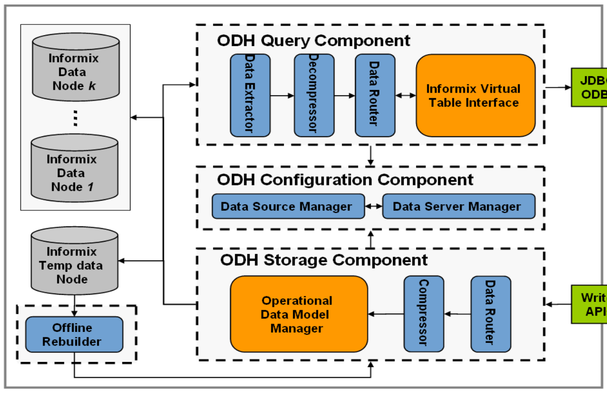


Figure 2. The ODH System Architecture

In practice, tags with different data characteristics are found in operational data from different data sources. The tags collected by high frequency sensors frequently fluctuate, while the tags collected by low frequency sensors are relatively stable. For those two different distributions of tags, the compressor uses a data variability-aware compression strategy that adopts different algorithms to compress the data. As shown in Figure 3, for smooth values of tags that are produced by low frequency sensors, the linear compression algorithm [7] is applied when the values are put into RTS or IRTS batch structures. The basic idea of linear compression is to represent multiple successive data values as a straight line that can be represented by its two spike points. For non-linear high-frequency tag values, the quantization algorithm [8] is applied, which builds a many-to-few mapping on the value ranges to decrease the number of bits that represent the values. The quantization algorithm can achieve 4-to-16-fold compression

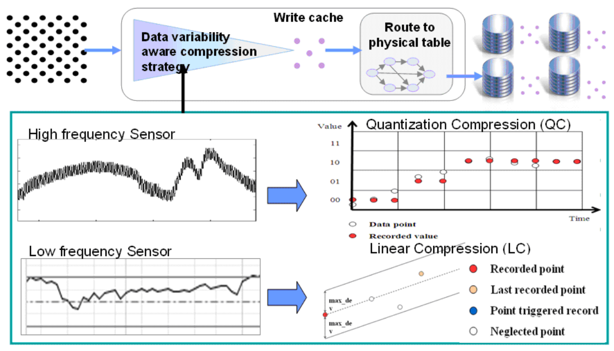


Figure 3. Compressed and Transparent data store

ratio, varying from the bits that are used to represent a data point. Both of the algorithms support lossless compression, as well as lossy compression with error bounds. Combined with the efficient data model, the ODH system can achieve about 10-to-100-fold data compression ratio with acceptable mean-square error bounds.

The ODH query component runs standard SQL queries and extracts the requested data that are distributed among different data servers. In IoTscenarios, the data is modified before being inserted. Therefore, the query component adopts a “dirty read” isolation level to access uncommitted rows from concurrent insertions. The power of SQL enables complicated analytics on both relational and operational data. The Informix Virtual Table Interface (VTI) [3] hides the details of the underlying infrastructure, the data distributions, and the different types of batch structures. VTI enables the operational data model to be accessed through virtual tables using standard SQL interfaces, which enables the fusion with other relational tables that are managed by Informix data servers. The operational records that share a same schema type can be accessed through a single virtual table (*id*, *timestamp*, *tags*). In ODH, a virtual table is generated and exposed to the users for each schema type. For example, environment monitoring sensors that produce (*timestamp, id, temperature, wind*) tuples can be shown in a virtual table named environ_data_v that is created with the same data schema. In this way, we enable users to select data from multiple data sources through one simple SQL query. For example, the following SQL statement returns the temperature and wind values from the environment monitoring sensors in area S1 from Nov. 18 to Nov. 22, 2013:

SELECT timestamp, temperature, wind FROM environ_data_v a, sensor_info b WHERE a.id=b.id AND b.area=’S1’ AND timestamp BETWEEN ’2013-11-18 00:00:00’ AND ’2013-11-22 23:59:59’

In this statement, the virtual table environ_data_v is queried along with the traditional relational table sensor_info. As the stored data are naturally partitioned by timestamps and data sources, we adopt a strategy similar to the partition elimination by filtering the timestamp and id provided in the WHERE clause so that irrelevant data nodes or tables are not scanned during a query. Another problem is efficiently processing queries that fuse operational data with relational data. For example, there are two possible query plans for the SQL statement above that fuses operational and relational data. One query plan extracts the records from the sensor_info table where the area field has a value of "S1" and then extracts the operational records from the environ_data_v virtual table that have timestamps between "2013-11-18 00:00:00’" and "2013-11-22 23:59:59" for each sensor id. The other query plan extracts the operational records in the environ_data_v virtual table that have qualifying timestamps and then performs a left join between each sensor id and its area from the sensor_info table. We designed a cost model for the operational data stored in different batch structures. Because the major performance blocker for queries is I/O, and the *ValueBlob*s in the batch structures create most of the I/O burden, we approximate the cost of extracting the requested operational data as the expected size, in bytes, of the *ValueBlob*s that need to be accessed. The estimated costs enable the Informix query optimizer to determine an optimal query path.

4. REAL-WORLD CASES

For asset-intensive industries like Energy and Utilities, connected vehicles, and transportation, the business performance is dependent on the deployed assets. In the past three years, we have deployed many systems for IoT data management across the heavy assets management domain. In this section, we briefly share our experiences of deploying ODH in real-world projects to address the pain points of big operational data management for real customers.

4.1 Power Grid for WAMS System

Power Grid A, one of the biggest power grid companies in the world, plans to conduct the world's largest Phasor Measurement Units (PMU) deployment, with more than two thousand PMU sensors that have a sampling frequency of 50Hz. The Wide Area Measurement System (WAMS) is designed to ensure power grid reliability and security by using PMUs to monitor the AC waveforms. The WAMS must process 100,000 incoming data points per second and real-time queries simultaneously. ODH is adopted as the real-time historian in the system.

The performance test that the customer required was successfully provided on an IBM P460 Server that was running four 8-core 64-bit 3.2GHz CPUs, with 128 GB of memory mapped to V7000 storage. We tested the performance of ODH under three different settings by simulating 2000 25 Hz PMUs, 3000 50 Hz PMUs, and 5000 50 Hz PMUs. Table 2 shows smooth CPU usage for all three cases. Note that for setting 3, we reduced the number of the CPU cores to 8 to obtain a reasonable CPU usage rate on a typical running server. Presumably, the average CPU load for setting 3 would be around 4.2% if the number of the CPU cores is increased to 32. Therefore, the CPU usage of the server depended almost linearly on the number of incoming data points per second, which were 50,000, 150,000, and 250,000, respectively.

Table 2. Performance Test on WAMS under different PMU Settings

#	*PMU Setting*	*# Cores*	*Avg CPU Load*	*Max. CPU Load*
1	2000@25 Hz	32	0.6%	1.7%
2	3000@50 Hz	32	2.2%	4.3%
3	5000@50 Hz	8	16.8%	25%

4.2 Smart Meters for AMI

Province Grid Power Company B in China wants to build a new Advanced Meter Infrastructure (AMI). Currently their system supports up to 3.6 million meters, but samples the data only once a day, due to real-time performance limitations. By 2015, the sampling rate must increase to every 15 minutes to conform to the national standard. Meanwhile, the number of smart meters will scale up by a factor of 10, which will result in a 1000-times expansion of data volume (from 7 TB/year in 2012 to 7 PB/year by 2015). Such an increase in volume is a big challenge for their current system, which is based on an relational database infrastructure.

The performance test of ODH for Company B was conducted to address this challenge. The test was performed with AIX 6.1 on an IBM P750 Server with a 16-core 64-bit 3.3 GHz CPU, 128 GB of memory, and 5 TB of XIV storage. We tested the performance of ODH over 30 days by simulating 35 million meters with a 15-minute sampling rate. ODH achieved very low CPU usage (<20%) and a memory usage of 80%. Most of the memory was used by the buffer pools of the MG batch structures and the Informix internal buffer pools. ODH achieved real-time data persistence with significant space saving for 35 million meters compared to a relational database. ODH also provided quick slice querying to enable real-time power consumption reporting. The data for 35 million meters with a 15-minute sampling interval was inserted into the database within 7 minutes. Each slice query for the 35 million meters took 150 to 200 seconds.

4.3 Connected Vehicles

A connected vehicle is a vehicle that is equipped with internet access, which allows information sharing with other smart devices both inside and outside of the vehicle. As the number of connected vehicles scales up to 1 million in three years, Company C's current data platform that uses a relational database has great challenges.

Our POC system with Company C shows the cost-effectiveness of using ODH as the real-time historian on a single IBM P750 Server with a 16-core 3.3GHz CPU, 128 GB of memory, and 2 TB of XIV storage. The performance test was conducted under three different settings by simulating 100,000, 200,000, and 300,000 vehicles. Table 3 shows the summary for the three cases. The results show that a single ODH server could support up to 2,500,000 connected vehicles that generate data in 10 second intervals. The increase of CPU load is mainly due to the increased number of threads we set for each workload which brings additional resource contention. In addition, a key factor for Company C is that they do not need to change their applications, which are built on the SQL interface, for the scale-up migration to ODH.

Table 3. ODH test for connected vehicles

#	*Vehicle Number*	*Avg Insert Throu.* (data points /s)	*Avg IO Throu.* (bytes /s)	*Avg CPU Load*	*Total number of MB written*
1	100,000	2.2M	18.04M	8.6%	368814
2	200,000	4.4M	36.08M	19.1%	740461
3	300,000	5.6M	45.92M	41.2%	946377

5. IOT-X: IOT DATA BENCHMARK

To the best of our knowledge, currently there is no gold standard for operational data management of IoT. The commercial nature of operational data of IoT makes it hard to find enough public operational data sets to construct a general benchmark. To give a comprehensive estimate of the performance of the ODH system and also to help the research community with a comprehensive picture of the IoT world, we designed the IoT-X, a benchmark that reflects aspects of operational data management in real-life IoT scenarios. The X in the name refers to the data seeds used in the benchmark. In the future, we plan to add more data seeds that are either derived from other benchmarks or from data seeds with real-world characteristics. Figure 4 shows the big operational data spectrum that we define, which is based on our engagement experience with real customers. Note that we do not consider data with a throughput that is lower than 100,000 data points per second to be big operational data because such data can be easily handled by traditional relational databases. To cover the spectrum of big operational data, we introduce two series of datasets in IoT-X. One series of datasets focuses on examining the performance of high-frequency big data, while the other series tests the performance of low-frequency big data.

IoT-X is divided into two workload suites, each of which tests one aspect of the performance on operational data ingestion and querying. We ran the benchmark against ODH, a popular commercial relational database, denoted as RDB, and the most-widely-used relational database, MySQL [4]. The results show that the write performance of ODH is higher than the relational databases by more than an order of magnitude. Also, the query capabilities of ODH are comparable to that of the relational databases.

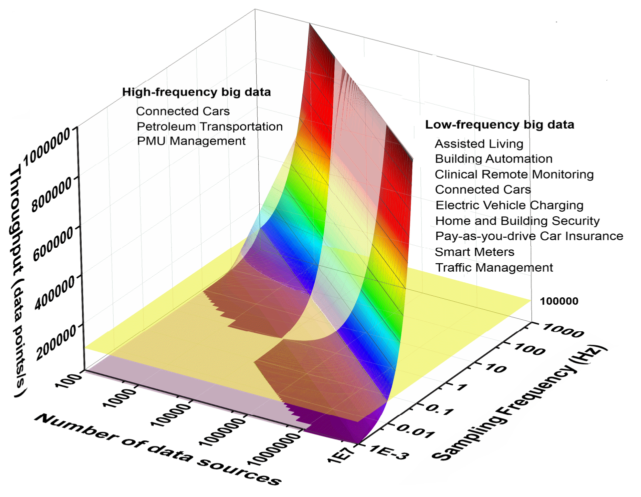


Figure 4. The Spectrum for Big Operational Data in IoT

5.1 Datasets

In this section, we describe the datasets used in IoT-X. For all the datasets, we divided the data into two categories: the operational data and the corresponding relational data. The operational data are the steadily growing records that contain timestamps for each record. The relational data typically contain information about the data sources and other business information. In the current benchmark, we used two data seeds. Based on the data seeds, two series of datasets were generated. The first data seed IoT-D$_{TPC-E,}$ (or TD for simplicity) was derived from the TPC-E benchmark dataset [9] to simulate high-frequency big data. The second data seed IoT-D$_{LSD,}$ (LD) was derived from the Linked Sensor Dataset [10], which is generated by the weather stations in the United States, to cover low-frequency big data. Table 4 summaries the datasets generated by the seeds in IoT-X.

Table 4 The datasets used by IoT-X

Dataset	*Data Seed*	*# of Data Sources*	*Avg. S. Freq. (Hz)*
TD(i, j) (i, j = 1, …,5)	TPC-E	i*1,000	j*20
LD(i) (i = 1, ..., 10)	LSD	i*1,000,000	1/23

The TPC Benchmark E (TPC-E) [7] is a well-known On-Line Transaction Processing (OLTP) benchmark for relational databases. We used a simplified version of this benchmark dataset as the initial data seed, which is composed of customer, account, and trade data. A customer has multiple accounts and the Trade table contains the trade records for all the accounts. There is an average of five accounts per customer. A detailed description of the dataset and the fields of different tables can be found in the TPC-E Standard Specification [9]. Specifically, TD implemented the three tables as follows:

Customer (C_ID, C_L_NAME, C_F_NAME, C_TIER, C_DOB);
Customer_Account (CA_ID, CA_C_ID, CA_NAME, CA_BAL);
Trade (T_DTS, T_CA_ID, T_TRADE_PRICE, T_CHRG, T_COMM, T_TAX) ;

In IoT-X, we considered accounts as the data sources. Each trade record in the Trade table is an operational data record. In addition to the account information, we also included customers as additional relational data. We removed some fields in the original Customer and Customer_Account tables from this simplified version because IoT-X mainly focuses on the operational data aspects. We used the TPC-E data generator (EGen) to generate five 2000-hour-long data seeds with different numbers of accounts (i.e. data sources), as shown in Table 4. By default, the EGen generates more than 1000 customers. We modified the value of the *load_unit* variable in the EGen code from 1000 to 200 to decrease the lower bound. Furthermore, because the highest overall trade frequency that the EGen can simulate is *load_unit* records per second, the data generated by EGen cannot be used directly as high-frequency datasets. Instead, we set the overall trade frequency for EGen to 100Hz to simulate high-frequency big data by increasing the speed of trade processes. For each of the data seeds, we shortened the intervals of the T_DTS timestamp fields to generate 5 datasets with different sampling frequencies, and truncated the datasets to one hour long. For example, for the dataset TD(1, 1), which has 1000 accounts (i.e. 200 customers) and an account trade frequency of 20 Hz, the expected throughput should be 20,000 trades per second, which results in a 200-fold increase in speed. To enable efficient querying, B-tree indices are created on T_DTS and T_CA_ID.

The Linked Sensor dataset contains extensive descriptions of weather in the United States in RDF form. A series of datasets are provided, each of which contains weather data for a storm. In IoT-X, we adopted the hurricane Ike dataset, which is the biggest dataset, as our data seed for the low-frequency datasets. The Ike dataset contains approximately 500 million triples (101 GB) from Sept. 1 to Sept. 13, 2008, generated by 12,336 weather sensors. The data can be assembled into approximately 10 million operational data records. The average sampling interval is approximately 23 minutes per record. In addition to the operational records extracted from the weather data, we also captured the information about the sensors into a relational table. A data adapter was developed to convert the RDF data into comma-separated value (CSV) files, which were consumed by the workloads. We implemented the two tables as follows:

LinkedSenor (SensorId, SensorName, Latitude, Longitude);
Observation (Timestamp, SensorId, WindDirection, AirTemperature, WindSpeed, WindGust, PrecipitationAccumulated, PrecipitationSmoothed, RelativeHumidity, DewPoint, PeakWindSpeed, PeakWindDirection, Visibility, Pressure, WaterTemperature, Precipitation, SoilTemperature);

Based on the data seed, we sped up the data rate by 60 times and generated 10 low-frequency datasets by scaling up the number of linked sensors from 1,000,000 to 10,000,000, with an increasing step of 1,000,000. Each dataset was truncated to two hours long. B-tree indices are created on *Timestamp* and *SensorId.* We did not simulate different sampling frequencies for the low-frequency data. From the experience of real projects, we found that small sampling frequencies have little direct impact on system performance. What really matters for low-frequency big data is the number of the data sources, or more precisely, the number of expected data points per second that reach the system. We argue that the ten datasets with different numbers of data sources cover the data characteristics well for low-frequency big data.

In the Linked Sensor Dataset, the operational data belonging to different linked sensors have different data schemas. We handled this problem by constructing the Observation table as a universal set of all possible measurements produced by different sensors. For example, the sensor named "A07" only measures WindDirection, AirTemperature, WindSpeed, and WindGust. All the other attributes are always NULL. This simple data schema is of practical significance because the sparseness of operational data records is very common in real-life applications. For example, in a smart grid application, a meter is usually able to generate operational records with 10-200 tags. When designing the data system, all the tags should be included. However, due to the technical limitations of meters and transmission, in practice, only tens of tags are collected by the meter, and all the other tags have the value of NULL.

Table 5 The query templates for TD

Templ. ID	***SQL templates***	***Comments***
Simple templates		
TQ_1	select * from TRADE where T_CA_ID=*id*	Historical query
TQ_2	select * from TRADE where T_DTS between t and $t+\Delta t$	Slice query
Complex templates		
TQ_3	select T_DTS, T_CHRG from TRADE t, ACCOUNT a where a.CA_ID = t.T_CA_ID and a.CA_NAME = *name*	Single data source involved
TQ_4	select CA_NAME, T_DTS, T_CHRG from TRADE t, ACCOUNT a, CUSTOMER c where a.CA_ID = t.T_CA_ID and a.CA_C_ID = c.C_ID and C_DOB between t_1 and t_2	Multiple data sources involved

Table 6 The query templates for LD

Templ. ID	***SQL templates***	***Comments***
Simple templates		
LQ_1	select * from Observation where SensorId = *id*	Historical query
LQ_2	select Timestamp, SensorId, AirTemerature from Observation where Timestamp between t and $t+\Delta t$	Slice query
Complex templates		
LQ_3	select Timestamp, SensorId, AirTemerature from Observation o, LinkedSensor l where l.SensorId = o.SensorId and SensorName = *name*	Single data source involved
LQ_4	select Timestamp, SensorId, AirTemerature from Observation o, LinkedSensor l where l.SensorId = o.SensorId and Latitude < la_1 and Latitude > la_2 and Longitude < lo_1 and Longitude > lo_2	Multiple data sources involved

5.2 The Workload Suites

The core of the IoT-X benchmark consists of two workload suites that exercise specific features of an operational data management system. Both of the workloads suites run a series of workloads on the two series of datasets.

WS1, Write: The write workload suite examines the real-time write performance of the operational data management system. A data simulator was developed to read data from standard CSV files and simulate real-time data insertion to different operational data management systems. Currently, the simulator supports two types of insert interfaces: the ODH Write Interface and the standard JDBC interface. For the TD datasets, we exported the operational data in the Trade table into CSV files and simulated the trade record insertions. 25 workloads were created from the 25 TD datasets for each testing candidate, with different combinations of account trading frequencies and numbers of accounts. For the Linked Sensor Datasets, we exported the operational data in the Observation table into CSV files and simulated the observation records in the 10 LD datasets with 10 workloads. The average throughput, maximum throughput, average CPU rate, maximum CPU rate, and the actual storage size were reported for each of the workload. Note that despite the known insert velocity of the simulator, the actual data throughput of the testing systems was not necessarily equal to the insert throughput, due to performance bottlenecks or system bursts.

To be fair to the relational databases that use JDBC, we disabled the autocommit feature of JDBC and used the batch insert mechanism to optimize the write performance. The simulator calls the *executeBatch* function for every 1000 operational records. Our experiment shows an average of a 10-fold increase in speed by using batch inserting.

WS2, Read: The read workload suite examines the SQL query capacity and the performance of the operational data management system. We designed a series of SQL query templates for both TD and LD to test the candidate systems with different types of queries. We classified the query templates into simple query templates and complex query templates, which cover most typical queries that we observe in IoT scenarios. The simple query templates generate queries for operational data, while the complex query templates fuse operational data and relational data. The datasets TD(5, 2) and LD(5) were used in this workload suite to test the query performance of high-frequency and low-frequency big data, respectively. During the data preparation, the operational data were inserted into the database by *WS1* while the corresponding relational data were manually imported into the system of the testing candidates in advance. The query templates are listed in table 5 and table 6.

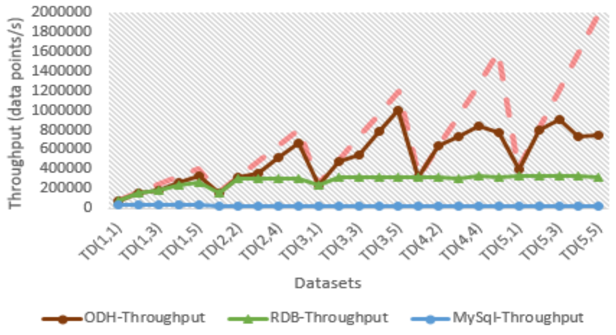
For each of the query templates, a workload was created to generate queries based on the template. Each workload randomly generated parameters for its template to convert the workload to 1,000 concrete queries that were run in order. For TQ_4 and LQ_4, Δt follows the uniform distribution valued from 1s to 10s. All the other parameters were picked randomly among the actual values in the datasets. The metrics that were reported by each workload include the average data throughput of the queries and the CPU usage.

5.3 Benchmark Test against ODH

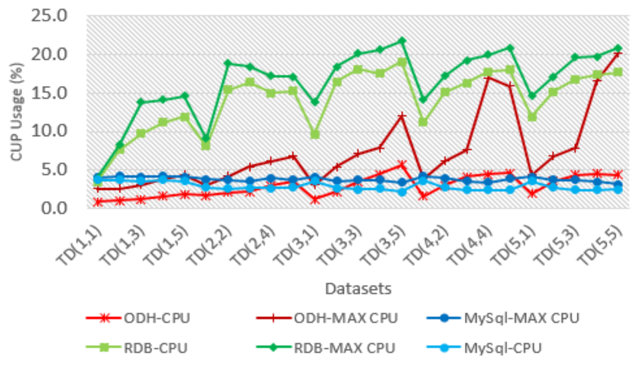
In this section, we report the IoT-X results of the ODH system, as well as the performance on relational tables in the RDB and MySQL. We conducted all experiments on an 8-core 4060 MHz Power PC with 32 GB memory and a RAID5 10 TB storage with 2 Gbps data bandwidth for both reading and writing.

Figures 5 and 6 show the insert performance of the three candidates for the various settings listed in Table 4. The red dashed line is the actual overall data throughput of the data sources. For both TD and LD datasets, the ODH system is superior in terms of CPU rates and data throughput. The performance upper bound is at around 1 million data points per second for TD datasets and 1.5 million data points per second for LD datasets. This rate of data throughput covers almost all the data scenarios that we have seen in real-world deployments. In comparison, both RDB and MySQL failed to insert the data in real-time for most of the datasets. The data throughput was much lower than the speed of data generation by the data sources, especially for MySQL. At times, the write performance of RDB and MySQL was just too low to insert all the data. In those cases, we forcedly terminated the unfinished writing processes after 4 hours.

Furthermore, we saw a gradual performance decrease for both the relational solutions during each run. This decrease happened because as the number of records increased, the cost of updating the B-Tree index increased dramatically and became a bottleneck for the whole system.

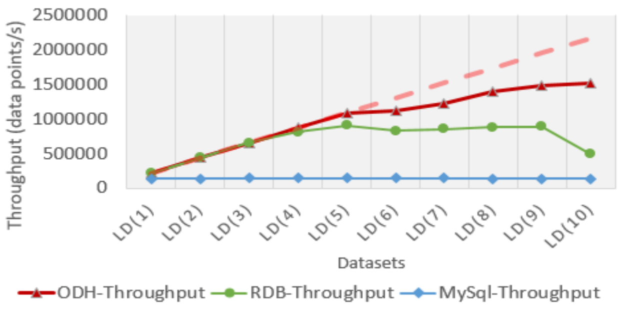


(a) Throughput (data points per second)

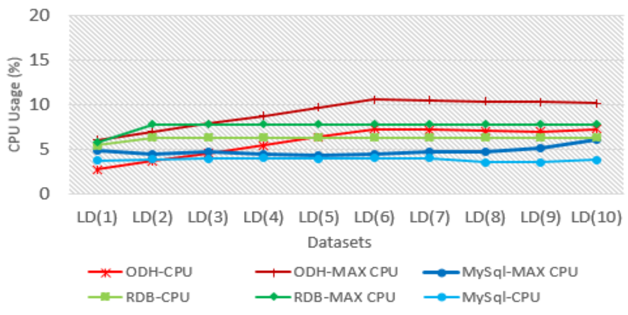


(b) CPU rate

Figure 5. Insert throughput and CPU rate for the TD datasets



(a) Throughput (data points per second)



(b) CPU rate

Figure 6. Insert throughput and CPU rate for the LD datasets

Both TD and LD generate irregular time series data. As a result, IRTS and MG batch structures are used for the two series of datasets, respectively. On average, RDB and MySQL always consumed almost the same amount of storage for the same datasets, while ODH had space requirements that were smaller by a factor of more than 3. Due to the page limitations, we list only a portion of results here in Table 7. Basically, the storage size increased linearly with the frequency and number of data sources.

Table 7 Storage Cost for Selected Datasets (in MB)

	TD(1,1)	TD(1,2)	TD(1,4)	TD(2,1)	LD(1)	LD(2)
ODH	2171	4344	8647	3630	27057	54327
RDB	7115	14272	28458	12915	47750	95603
MySQL	7393	14748	29361	13304	50191	102164

Because our data compression features rely highly on the characteristics of incoming data and the compression parameters, we did not include a compression test in the benchmark. However, according to our rough test, applying linear compression on LD(1) with a maximum deviation of 0.1 from the original value, for example, led to a storage size of 1360 MB, resulting an overall compression factor of more than 35 compared to the sizes produced by the relational databases.

RDB performed surprisingly well on the LD datasets, and reached a data throughput with up to 1 million data points per second. The reason is that the large size (86 bytes) of each record dramatically reduced the magnetic arm movements, which is one of the major bottlenecks for RDB. However, in some IoT applications, such as Clinical Remote Monitoring and Traffic Management, the number of tags is typically no more than 5, which results in a small record size. To investigate the performance impact of the record size, we varied the number of tags of LD(10) from 1 to 15, and measured the write throughput of ODH and RDB for each size.

Figure 7 shows the result. The performance of RDB depended on the number of tags, while the performance of ODH remained high even when the number of tags was 1. Another important factor of the high performance is that with better compression, ODH does less I/O.

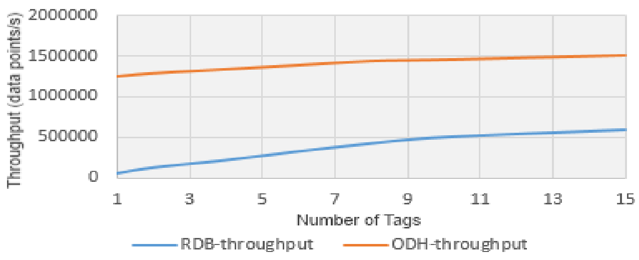


Figure 7. The number of tags vs. data throughput for LD(10)

Table 8 shows the query performance of the three candidates for the two datasets TD(5, 2) and LD(5), which are in the middle of the IoT operational data spectrum. For each query template, 100 queries were generated and the average data throughput and CPU rates were recorded. As shown in the table, the throughput of ODH is lower than the two relational databases. We profiled the whole query process for ODH and found that there are two main performance blockers. The first blocker is the data router. For each query, the data router looks up the metadata to locate the required date. This process is currently completed by SQL statements. This is the main reason of the low performance of LQ_1. Because of the low sampling frequency of the LD datasets, the number of the rows in each query instance of LQ_1 is no more than 100. Actually, the average execution time of each query instance of LQ_1 for all three candidates is lower than 100 ms. In this case, the overhead of data router dominates the whole query process for ODH. The second performance blocker is the cost to assemble the relational records in the virtual tables. A more than 80% overhead is observed during the data extraction process. That is to say, the throughput of TQ_2, for example, could reach 1.3 million data points per seconds if we bypass the SQL interface and expose the query component through Java or C APIs. This performance issue will be fixed in a future version of Informix.

Table 8 Query performance for the three candidates

Query	*ODH*		*RDB*		*MySQL*	
	Throu (dp/s)	CPU (%)	Throu (dp/s)	CPU (%)	Throu (dp/s)	CPU (%)
TQ_1	272340	3.4	543574	3.6	633568	3.4
TQ_2	271602	3.8	760559	4.54	725868	3.78
TQ_3	187385	3.3	167221	3.4	206111	3.5
TQ_4	177157	3.4	163875	3.6	214880	3.6
LQ_1	2640	3.2	303750	3.4	165000	3.52
LQ_2	10055	3.3	19289	3.54	30441	3.4
LQ_3	1571	2.2	18225	1.4	28969	3.63
LQ_4	8730	3.6	9225	3.2	6375	3.1

Because of the tag-oriented data model, we observed a performance boost for ODH compared with the other two candidates for TQ_3, TQ_4, and LQ_4, when one of the tags was extracted and the impact from the data router was very small.

To test the query optimizer, we constructed a series of LQ_4 queries and logged the query plans. The results showed that the query optimizer gave reasonable cost estimations. For example, consider the query plan of the query (la1=36.803; la2=36.804; lo1=-115.978; lo2=-115.977). The specified area involves only one sensor. In this case, ODH first located the sensor in the LinkedSensor table and then extracted the data from the Observation table for the sensor. In contrast, the query plan of another query (la_1=10; la_2=80; lo_1=-150; lo_2=-50) which involved a large number of sensors, first selected all the data from the Observation table and the performed a left join of the location information in the LinkedSensor table to the results.

Lessons Learned. (1) When the write throughput is lower than 500 K data points/s (if a machine with lower end hardware configuration is used, this threshold is even lower, down to 100 K data points/s), the commercial relational databases are competent to manage all the data. (2) Based on our experience in engagements with the real customers, ODH with the hardware environment we used in the benchmark test can cover most of the requirements for real-time data ingestion. (3) ODH has better tolerance to ingestion fatigue than relational databases, because relational databases require a B-Tree update for each record insert. (3) The smaller the size of an operational record produced by an individual sensor is, the larger the write performance gap is from relational databases to ODH. (4) ODH saves a lot of storage compared to the relational databases through efficient data structures and IoT data-oriented compression algorithms. (5) The query performance of ODH is already good enough to our customers. Still, due to the VTI overhead of ODH, relational databases perform better for many queries. To solve this issue, the efficiency of Informix VTI needs to be improved.

6. SUMMARY

In this paper, we present a new operational data historian that extends the Informix database server to address the problem of volume and velocity of event/motion data management in IoT applications. To meet the high velocity data write requirement, the system packages and compresses time series data into its own data structures. The optimized writing technology improves the write performance by a 1-2 order of magnitude compared with a traditional relational database. The unique TSDB/relational database fusion through standard SQL interfaces allows the querying and analyzing of data in IoT applications with little impact on the original programming model. We also share our experiences of operational data management from real-world cases. Finally, we present the first IoT data benchmark, IoT-X, and provide a comprehensive evaluation of the data management systems, including ODH. We plan to contribute IoT-X to the open source community so that other vendors can evaluate their data management systems for the big operational data historian requirements of IoT. In addition, we also plan to add more real datasets to the benchmark to represent different IoT industries. Future work for the ODH system includes adding proper indexing to reduce BLOB scanning for queries on attribute values, improving query performance, and integrating advanced data analytics functionalities.

7. REFERENCE

[1] Azza A., Kamil B., Daniel A., Avi S., and Alexander R. 2009. HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads. *Proc. VLDB Endow.* 2, 1 (August 2009), 922-933.

[2] Andrew P., Erik P., Alexander R., Daniel J. A., David J. D., Samuel M., and Michael S. 2009. A comparison of approaches to large-scale data analysis. In *Proceedings of the 2009 ACM SIGMOD* (SIGMOD '09), Carsten Binnig and Benoit Dageville (Eds.). ACM, New York, NY, USA, 165-178. DOI= http://doi.acm.org/10.1145/1559845.1559865

[3] Informix Virtual Table Interface. http://www-01.ibm.com/support/knowledgecenter/SSGU8G_12.1.0/com.ibm.vti.doc/vti.htm

[4] MySQL. http://www.mySQL.com

[5] PI Server. http://www.osisoft.com

[6] Write paper. http://www.oracle.com/us/industries/utilities/ultilities-exadata-exalogic-wp-1499854.pdf

[7] Hale, J. C.; Sellars, H. L. Historical Data Recording For Process Computers. Chem. Eng. Prog. 1981, 77 (11), 38-43.

[8] Quantization. http://en.wikipedia.org/wiki/Quantization_(signal_processing)

[9] The TPC Benchmark E. http://www.tpc.org/tpce.

[10] H. Patni, C. Henson, and A. Sheth. Linked sensor data. 2010.

[11] TempoDB. http://tempo-db.com/