Sensor Data Storage for Industrial IoT

Michael Donaghy, Global Business Development Executive

Contents

Introduction

Businesses adopting Industrial Internet of Things (IIoT) technologies face a herculean task. In an industry where the number of connected devices and equipment is increasing exponentially, the amount of operational data coming online which must be analyzed and stored rises in tandem. Though many industrial businesses utilizing traditional database management systems find themselves drowning in the sheer volume of sensor data they are producing, those who are forward-looking are turning towards new platforms to help manage their data and, in turn, reduce costs and improve overall performance.

This paper traces the lifecycle of Industrial IoT sensor data from its initial point of entry into a system, through various transformations and tiers of storage, to finally its archival or when disposition.

As a device or sensor generates data, it needs to be ingested and stored where it needs to be analyzed. This is typically close at the physical location of the sensor or device as well as at central facility.

Data first is ingested into an in-memory database where it is processed to perform validations or checks, identify patterns or trends, detect anomalies, and make decisions on this fresh data. As the volumes of data and the richness of processing increase, the amount of available memory can quickly be exhausted.

To optimize the amount of space available in-memory, data can be filtered, that is, it can have any redundancies or noise removed, either at the point of ingestion or just prior to the data being stored to a permanent storage medium.

As data is moved from memory to disk, it can be placed on storage media based on performance and cost criteria, where it can then be compressed to reduce the amount of data which is stored. When the high-precision raw sensor data you have stored has reached the end of its usefulness, you can aggregate or "downsample" that data so that it is reduced in size whilst still maintaining various statistical measures.

These techniques, their implications, and how they function within Kx are explained in greater detail in the sections that follow.

In-Memory Data Optimization

During the first stage of its lifecycle, data is ingested into an in-memory database where proactive analysis can be performed upon it. With Kx, data which is ingested at the head-end is stored in its real-time (in-memory) database.

With financial trading systems, from where Kx has its roots, this in-memory database (often called a real-time database) holds an entire day's worth of trades, allowing analytics to be performed on the data whilst it is in-situ. When the day's trading comes to a close, this data can simply be stored on disk, the in-memory database can be cleared and begin anew the following day.

In the world of industrial IoT, the approach is often quite different. And this difference is mostly down to the fact that the sheer volume of data which manufacturers create in a day through ingestion from tags, sensors and other connected devices dwarfs that of the financial trading world. The storage of this data in-memory has far-reaching implications for those who operate in industrial IoT. They may want to use in-memory data to perform complex event processing (CEP) to identify trends and issues of concern and then load new scripts into devices to change their behaviour.

To exemplify the early identification of trends or issues, consider a transformer in an electrical grid. These transformers are used to change electricity into different voltages to fit necessary conditions. They generate an inordinate amount of heat performing this process and therefore need to be kept cool with insulating oil. The temperature of this oil is of critical importance. The transformer will have a number of sensors measuring oil temperature, and this information will be fed into the real-time database. If the temperature of the oil increases to a concerning degree, this trend needs to be identified promptly so that the system can determine whether or not the load needs to be changed or perhaps the transformer is undersized for its load.

Now to put the scale of this data into perspective, some manufacturing execution systems can receive around 4.5 million data points per second from their sensors. This volume, incidentally, is something on which Kx can easily run multi-variate analysis or apply machine learning models, allowing for the detection of trends and ensuring machine behavior can be changed in a proactive manner.

Nonetheless, you must deliberate whether or not storing the entirety of this ingested data in-memory is completely necessary. When operating at high velocities – where not all data can be kept in-memory – it is vital to have a very efficient way both of sending data to disk as well as retrieving it from disk, whilst simultaneously supporting ingestion and analytics.

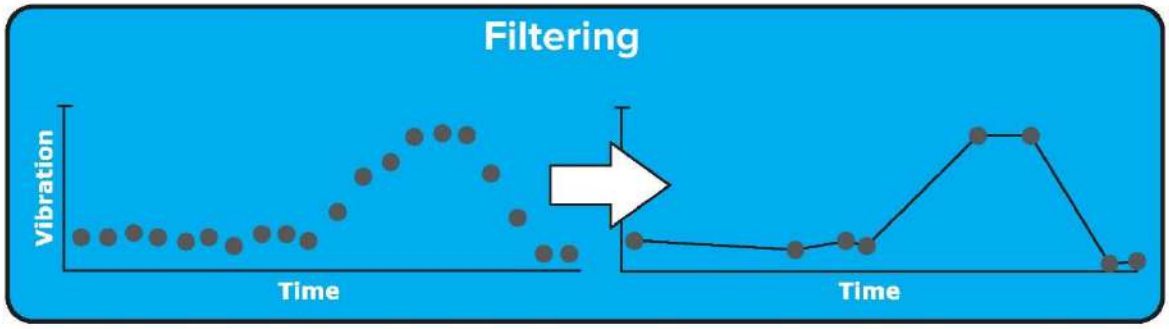
Filtering

Not all data is of equal value. Where some business may only care for statistical summaries others will require full and exhaustive records of all measurements.

There is a consideration that must be made in regards to maximizing the amount of useful data which is stored in memory and minimizing the amount of 'noise' or redundant repeating data that is fed into this stage of the data storage lifecycle.

One technique to manage what data is retained while maximizing its value is called filtering, a form of "lossy compression". Filtering refers to removing redundant data from a data stream or adjusting its precision while maintaining its usefulness for analytics and decision making.

A good way of explaining filtering in practice would be to think of an industrial welding robot. The tool the robot is operating with will likely have sensors measuring its vibration. These measurements should remain relatively static but might increase and decrease in each reading by a mere 0.1%. It would be wasteful to keep all of this data in storage when there has been no real substantial change. So instead of putting it in-memory or writing it on to a disk, we can run an algorithm over it to filter what's useful and what isn't, thus discarding the intermediate data points which are basically identical.



With Kx, a variety of algorithms are available to filter out superfluous points from time-series data including the well-known Swinging Door as well as our own in-house variant, the Jacobian Filter. Nevertheless, one must evaluate all of the implications of filtering before deciding on how they frame their approach to the matter. Once data is gone it is gone forever. The irreversibility of the process means that a user doesn't want to risk throwing away data that may later turn out to have been useful. This can happen through setting poorly chosen parameters on the filter.

Pharmaceutical companies may need to retain and store all of the data they create due to regulatory requirements for their particular industry, so filtering is not employed. However, for most manufacturers, this

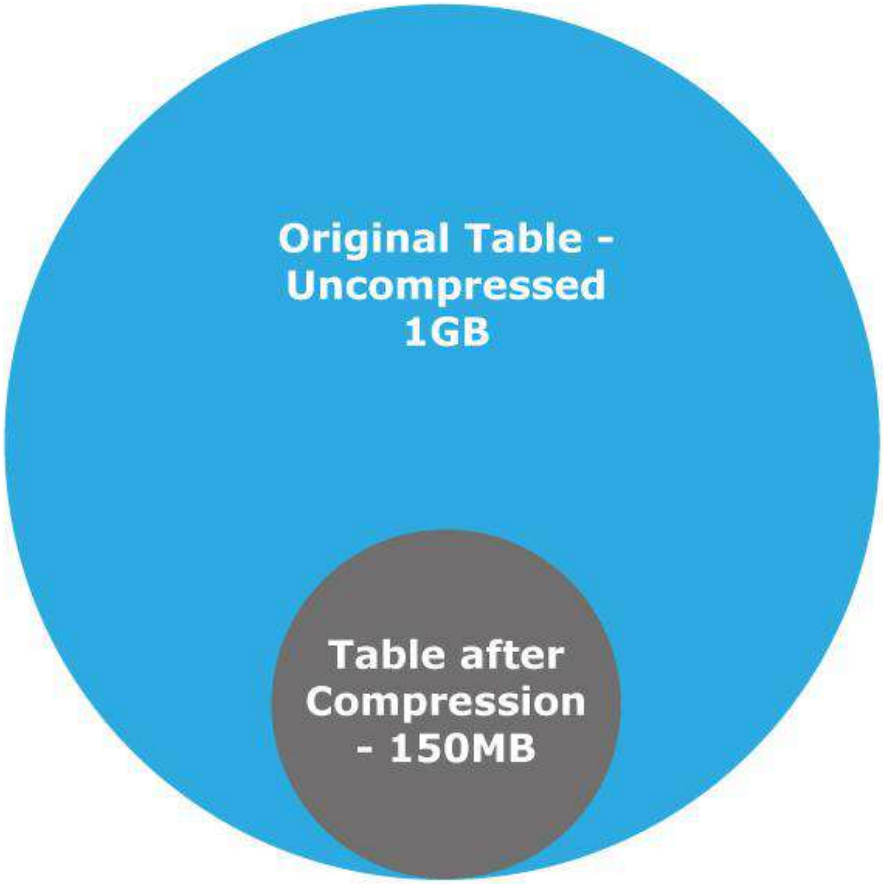
will not be needed and instead they should look to filtering out any data that is repeating, redundant and unnecessary.

In addition, a decision must be made on where sensor data filtering occurs. In order to maximize your in-memory database, you may want to filter data where it is generated or ingested. Doing this will certainly reduce the memory requirement needed but will mean you are throwing away a lot of early data and are therefore reducing your ability to manipulate the data as a whole when it is in-memory. Moreover, filtering at the head-end eliminates the option, available only when the data has been written on to storage media, of running several filtering algorithms comparatively in order to work out which gives the best data reduction.

Compression

Compression and filtering both have the same objective in that their goal is to reduce the amount of data that needs to be stored, thus maximizing the value of the data and optimizing the cost of storage management. Compression does this through looking for the best ways of packing data in to as little space as possible without losing any, therefore it is aptly referred to as being 'lossless'. Kx offers the advantage in that it allows compression to be applied to a table or varying compression to be applied to individual columns.

Kx comes with several built-in ways to compress data, each with differing performance and compression characteristics; this is enabled by its columnar orientation where "like" data can be compressed the most efficiently. A sample of the power of Kx in this regard can be found in the result of a recent test with the sample data of a prominent precision manufacturer in which Kx was able to take a table of 1GB and compress it to 150MB – 15% of its original size. However, this is only but a sample of our findings, and we recommend that you assess this with your own data.



Kx offers three software compression algorithms: Snappy, IPC and gzip. Each of these algorithms allows the user to configure different parameters in order to adjust the aggressiveness of compression. These parameters will have performance implications, for example, a more aggressive compression may take longer to perform, thus impacting performance. In conjunction with the three software compression algorithms, Kx also allows for hardware compression to be carried out and supports well-known hardware compression boards. Though these boards tend to be more costly, they can carry out compression at 40 times the speed of a software compression algorithm and therefore are incredibly useful when the speed of compression is a critical factor.

It is also worthy of note that Kx offers public utilities which can comparatively run through the varied permutations of the three aforementioned software compression algorithms and their parameters on your data to determine which provide it with optimal compression.

The Kx website has an interesting technical whitepaper entitled "DB Compression in kdb+" that discusses the performance of compression on queries.

Downsampling or Aggregation

Downsampling, or aggregation, refers to the process of taking a table of data, for example sensor readings, and attempting to aggregate or sample it within certain periodicities such as to turn it from a complete and comprehensive set to more of a statistic summary, showing only "highlights" of information such as the mean, standard deviation and various quantiles.

Downsampling is carried out when data reaches the near end of its lifecycle, when it is old and not nearly as useful as it once was. Rather, downsampled data, which can be accessed by Kx just like any other stored data, can prove convenient in the manufacturing industry when a failure occurs in the field. For example, in the event of a disk drive failure, and one wants to view the characteristics of the sensor data for that batch of disk drives (which may have been old enough to warrant downsampling) to check for any patterns between this batch and others. If one was to simply throw this granular data away, you would not be able to carry out this sort of analysis.

Downsampling provides a major advantage in data storage management. In the same tests carried out on the precision manufacturer with compression, Kx was used to downsample a table of sensor readings (100 millisecond intervals, sized at roughly 30GB). By only retaining readings at a 5 second interval, Kx was able to correspondingly reduce the table size to 600MB. Furthermore, compression was subsequently applied and reduced storage to 100MB – resulting in an overall storage being .3% of the original. The table below shows downsampling results in 1 second buckets from Kx.

Using Kx's powerful q-sql query language, Kx is able to access and combine downsampled data with the more recent high-precision data that is stored in other partitions.

							Percentile				
Time	Count	Mean	SD	Min	Max	Last	50%	75%	90%	95%	99.9%
00:00:01	2,249	101.3202	3.899206	87.36407	115.0097	100.8338	101.3106	104.0015	106.3257	107.5715	110.5468
00:00:02	2,288	101.3325	4.022687	87.84209	115.2454	104.4650	101.3138	103.9753	106.5695	108.0671	110.7447
00:00:03	2,288	101.3712	3.871499	89.55878	113.4745	98.08425	101.3478	104.1476	106.3581	107.7808	109.8663
00:00:04	2,289	101.3207	3.933148	87.71249	116.2347	102.7736	101.3121	103.8171	106.3512	107.7183	111.0315
00:00:05	2,288	101.2323	3.850531	85.43555	114.6439	98.62076	101.3632	103.9032	106.1060	107.3901	110.2334

Tiered Storage

Tiered storage refers to storing data on storage devices and media which commensurate with the performance, availability, and value of the underlying data, with the goal of optimizing your overall total cost of storage.

Typically, the newest, and likely more frequently accessed, data is placed on more expensive disk drives, such as SSDs, whilst older, less frequently accessed, data is moved to cheaper, lower performance storage devices. Furthermore, when data is mutating or in flux, it is important to store it in a faster medium.

Kx can leverage tiered storage through the use of partitioning and storing data as files on the filesystem. This enables data sets that are older or meet certain other conditions to be moved from the highest cost and highest performance storage to lower cost, lower performance storage over time. This provides great flexibility in choosing how data progresses along the storage lifecycle, allowing critical data to be retained on more high-performing disk drives.

Kx offers a performance advantage here. Due to the fact the analytics engine is integrated with kdb+, that is, data ingestion, analytics, and storage happen on the same hardware unit, the need to move data from one piece of hardware to another over a network is minimized. Thus analytics can be performed at the speed of the hardware they reside on, which is orders of magnitude faster than shipping them across a network.