IoT時代におけるロバスト性の高いヒストリアンの開発

Development of a Robust and IoT-Ready Data Historian

アズビル株式会社
ITソリューション本部
棚澤　武志
Takeshi Kurumisawa

アズビル株式会社
業務システム部
菊地　健一
Kenichi Kikuchi

キーワード
ビッグデータ，IoT，ヒストリアン，NoSQL，OPC

あらゆるものがインターネットにつながるIoT（Internet of Things）時代を迎え，これまでセンサデータの収集と蓄積に用いられてきたヒストリアンがより重要な位置を占めるようになると考えられる。これまでのセンサデータの蓄積のみならず，より高度な実績データの生成機能を実現するとともに，よりロバスト性を向上させるためのデータ冗長機能，経路冗長機能を備えたヒストリアンePREXION™を開発した。

As the world enters the era of the IoT (Internet of Things), sensor data collection and the data historians will have a more important role. We have developed a data historian known as ePREXION. In addition to simply collecting sensor data, it can produce results of a higher quality than existing systems and is equipped with data and circuit redundancies for improved robustness.

1. はじめに

あらゆるものがインターネットにつながり，あらゆる情報がコンピュータに取り込まれる時代となり，様々なセンサ，制御機器，プロトコルが開発され大きな技術変革を迎えようとしている。あらゆる情報がコンピュータに取り込まれる世界では，ビッグデータに対応できるデータベースが産業界では登場しているが，そのためには収集速度，データの保管方法，データの検索方法，データの解析方法など様々な技術課題が存在する。当社ではこれらの課題をいち早く捉え，センサデータをはじめとするヒストリデータのビッグデータ化に対応するため，新たなヒストリアンとしてePREXIONを開発した。

2. ePREXION のアーキテクチャ

2.1 ePREXIONのシステム構成

ePREXIONは，様々な分野におけるプラントや工場のDCSやPLCなどの監視･制御システムから製造データを自動収集もしくは他のシステムから書き込みし，蓄積，演算し，長期にわたりヒストリデータとして管理するシステムである。

図1　システム構成例

図2に示すようにePREXIONサーバは，リアルタイムヒストリサーバ，デバイスIOサービス，ヒストリデータベースから構成される。

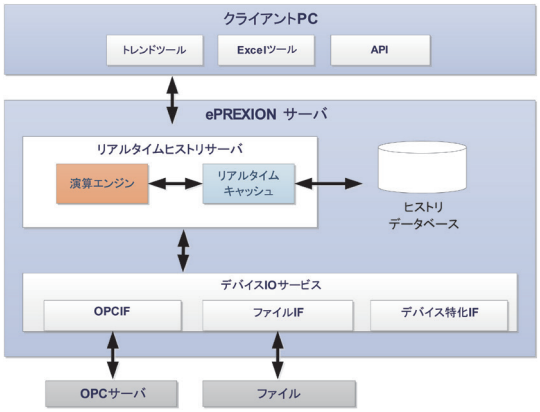

図2　機能構成

リアルタイムヒストリサーバは，クライアントアクセスを管理し，リアルタイムな応答性を高めるためのリアルタイムキャッシュ，演算を実行する演算エンジンから構成され，ヒストリデータベースの管理を行う。デバイスIOサービスは，標準ではDCS，PLCなどと接続するためのOPCインタフェース，他社システムとファイル交換するためのFileインタフェースやPLCと高速に通信するためのインタフェースを備える。その他のシステム接続にも対応するために，デバイスIOサービスはプラグインにより拡張可能となっている。クライアントからアクセスするためのAPIは，Microsoft®.NET Framework，OPC DAなどを提供しており，アプリケーションの構築に必要な環境を備えている。またデータ解析用のツールとして，ヒストリデータをグラフ化するトレンドツール（図3）と，Excelにてレポートを作成するツール（図4）を提供している。

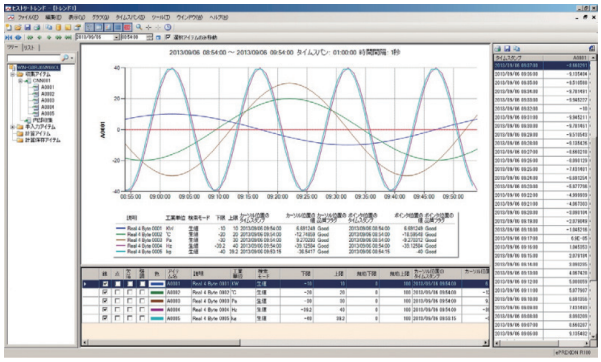

図3　トレンドツール

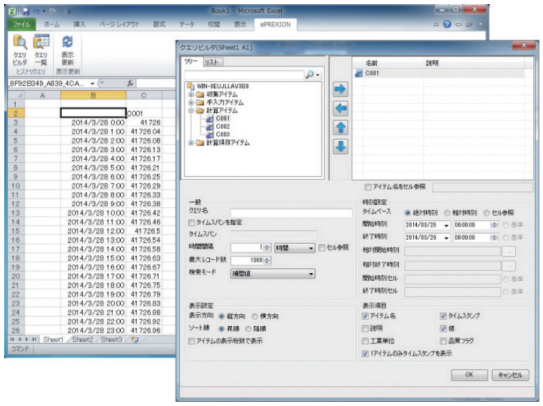

図4　EXCELツール

2.2　ePREXIONのデータベース

これまで様々なデータの格納基盤としてリレーショナルデータベースが多く用いられてきたが，パフォーマンスやスケーラビリティに多くの課題が存在する。そのためリレーショナルデータベースに代わりビッグデータを支えるデータベースとしてNoSQLと呼ばれるデータベースが産業界で登場しており，インターネットのサービスを支えるデータベースとして活用されている。NoSQLは従来のリレーショナルデータベースの課題を解決するための用途に最適化されたデータベースである。ePREXIONのデータベースもNoSQLと呼ばれるデータベースの1つであり，当社が独自に開発したデータベースエンジンを内蔵している。このデータベースエンジンは時系列のヒストリデータに最適化して開発したものである。

図5に示すようにリレーショナルデータベースを使用してヒストリデータを格納する場合は，一般的には時刻，センサ値を列として定義し，行として各時刻の値を格納する表形式でデータを保持する。

時刻	センサ値A	センサ値B	センサ値C
10:00:00	23.2	23.5	23.4
10:11:00	23.3	23.6	23.3
10:12:00	23.2	23.5	23.4
10:13:00	23.2	23.5	23.4
10:14:00	23.2	23.4	23.2

図5　リレーショナルデータベースでのデータ格納

リレーショナルデータベースを用いる場合は，同一タイミングで取得した複数のセンサ値を1つのテーブルにまとめて格納する。しかしながら1つのテーブルに複数のセンサの値を格納するとセンサの収集周期の変更を行った場合，テーブルごとに収集周期を変更する必要があるため柔軟性が低下するという課題が生まれる。ヒストリデータをリレーショナルデータベースに格納する製品では，長い期間のデータの分析や周期の短いデータの収集を行うと，ディスクに保存された多くのデータをスキャンする必要があるため，ヒストリデータの分析時の応答時間がデータの増加率以上に悪化する。

ePREXIONのデータベースは，一般的にKVS（Key Value Store）と呼ばれるNoSQLデータベースの構造を有している。図6に示すようにヒストリデータは項目ごとに「キー」と「値」のペアの配列として格納されており，「キー」として時刻を，「値」としてセンサ値を格納している。

センサA

時刻	値
10:00:00	23.2
10:11:00	23.3
10:12:00	23.2
10:13:00	23.2
10:14:00	23.2

センサB

時刻	値
10:00:00	23.5
10:11:00	23.6
10:12:00	23.5
10:13:00	23.5
10:14:00	23.4

センサC

時刻	値
10:00:00	23.4
10:11:00	23.3
10:12:00	23.4
10:13:00	23.4
10:14:00	23.2

図6　KVSでのデータのデータ格納

ePREXIONでは，キーとして時刻を値ごとに持つことになるため，単純には領域の効率が悪化すると考えられる

が，実際にはデータを格納する領域は少なくて済む。ヒストリデータの時刻は周期性を持つことや，センサデータは連続値をとる頻度が高いことから，非常に効率の高い圧縮アルゴリズムを適用できる。これによりセンサデータごとにディスクに保存する容量が小さくなり，ディスクへの読み込み範囲が少なくなることで高い応答性を実現している。また時刻を個々のセンサ値に持つことができるため，センサごとの収集周期を自由に変更できる。

2.3　ePREXIONの演算エンジン

センサから収集したデータを活用するためには，何らかの演算が必要である。これまでのヒストリアンでも様々な演算機能が実現されてきた。当社のヒストリアンでもセンサから収集したデータの演算機能を有していたが，様々な要件には対応できず外部プログラムを作成する必要があった。

ePREXIONでは，センサから収集したセンサタグ，任意のパラメータ値をヒストリ値として保持する手入力データタグ，並びにユーザーが任意に作成可能な関数の3つを計算タグの式として使用可能である。計算タグは呼び出し時に計算され，データが保存されないタグである。この計算タグ自身も演算式中に使用できる。呼び出し頻度の高い計算タグの参照時のパフォーマンスを向上させるため，計算タグの値を任意の周期で計算保存タグとして保存することができ，計算保存タグもまた演算式で使用できる。これらの関係を図7に示す。

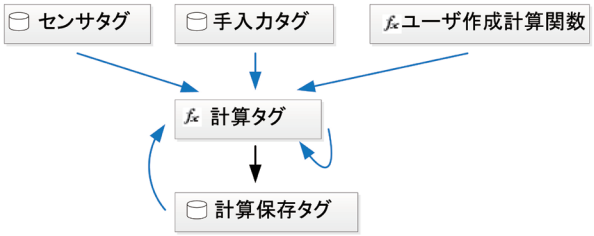

図7　演算機能の関係図

演算式では四則演算の他に，期間を指定した平均，合計をはじめとする集合演算や，条件式の記述ができる。これによりヒストリデータを用いた様々な計算が実現でき，あるケースでは外部プログラムによる計算をすべて置き換えることができた。

ePREXIONでは，これらの機能を実現するための機能とヒストリの演算に適した式の記述，実行エンジンを開発した。

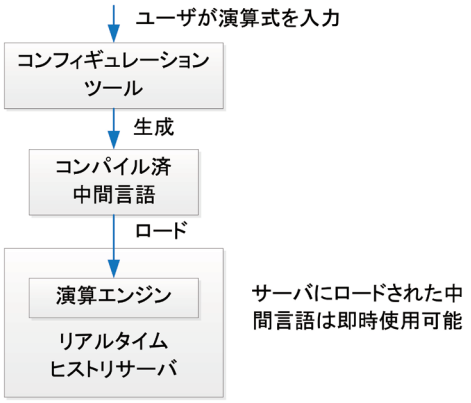

図8　演算エンジンのフロー

図8に示すようにユーザーが入力した式はコンパイルされ，コンパクトな中間言語が生成される。生成された中間言語は，リアルタイムヒストリサーバにて即時使用可能となる。コンパイル済の形式とすることで，動作の高速化を実現している。

図9に示すように計算タグの演算結果を保存する計算保存タグは，さらに別の計算タグで使用され計算保存タグとして保存するネスト構成が可能である。例えば各フロアのメーターから入力したデータを基に，フロアのエネルギー使用量を計算し，フロアのエネルギー使用量から事業所全体のエネルギー使用量を計算するなどの，階層的な計算に利用可能である。

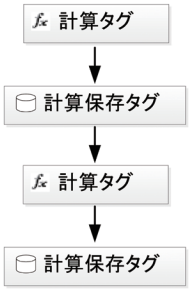

図9　演算機能のネスト

ネスト構成とした場合は，演算式で使用される元のデータが保存された後に演算を実行しないと正しい結果とならないため，依存関係に従い順番を設定する必要がある。しかしながらネストの深さが10階層などと多くなると，手作業で順番を付けることはコストのかかる作業であり，誤りも多くなる。そのためePREXIONでは計算タグと計算保存タグの使用関係を解析し，自動で順番を設定することを可能とした。

3.　ePREXION のロバスト性向上機能

ePREXIONでは，ヒストリアンのロバスト性向上のために，次の機能を提供している。

・ヒストリデータを保護するデータ冗長機能
・データ収集を行う経路を冗長する経路機能

3.1　ヒストリデータを保護するデータ冗長機能

従来ヒストリアンではサーバが故障した場合に備え，2つのサーバ上にヒストリアンを設置することによりサーバの故障に備えていた。しかしながらこの方式では，2つのサーバをコンフィギュレーションしなければならない，また2つのサーバのヒストリデータはタイミングや通信の遅延などにより差異が生じ，同じデータにはならないという課題があった。ePREXIONではコンフィギュレーションとヒストリデータの同時性を担保するための機能としてデータ冗長機能を開発した。

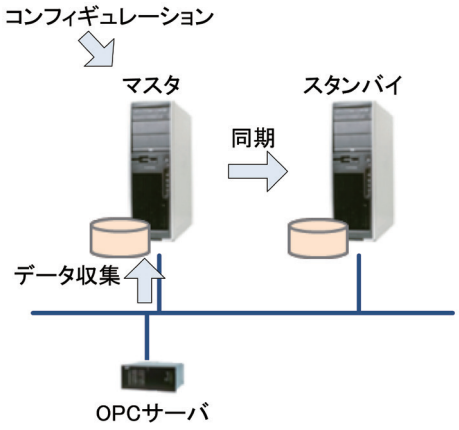

図10　データ冗長構成

図10に示すようにデータ冗長構成は，マスタとスタンバイの2つの役割のサーバで構成される。コンフィギュレーションはマスタサーバに対してのみ行うことが可能で，コンフィギュレーションしたデータは自動的にスタンバイに転送される。またデータ収集はマスタサーバのみが行い，収集されたデータはスタンバイに転送され反映される。このようにマスタとスタンバイのデータは常に同じとなる。マスタからスタンバイにデータが反映される時間は最大負荷がかかっている場合でも1秒から3秒程度と短い。またスタンバイのサーバは，トレンドツールやExcelツールなどからは読み込み専用のサーバとして使用できるため，正しくデータが同期されていることの確認や，スタンバイサーバを読込みの負荷の分散にも利用することができる。

マスタサーバが異常停止した場合やネットワークが切断された場合は，スタンバイサーバの役割がマスタとなり，データ収集が実行される。役割の切替えは手動でも行うことが可能である。

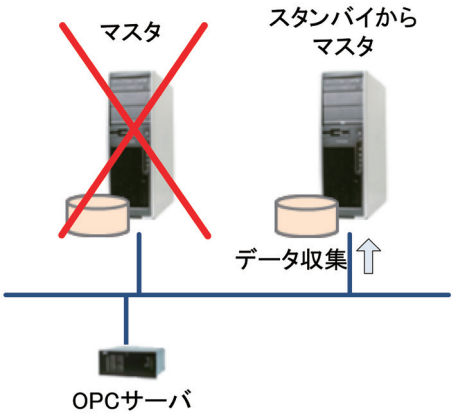

図11　データ冗長時の切替え

ePREXIONでのデータ冗長を実現するための方法としては，差分を転送する方法を用いている。図12にてデータ収集されたデータがマスタサーバからスタンバイサーバに転送されるまでの流れを説明する。

1. データ収集
   デバイスIOサービスでデータを収集する。
2. 差分転送
   デバイスIOサービスからリアルタイムヒストリサーバに収集データは送信され、さらにスタンバイサーバに転送される。
3. 差分適用
   スタンバイサーバでは転送された差分をヒストリデータベースに適用する。

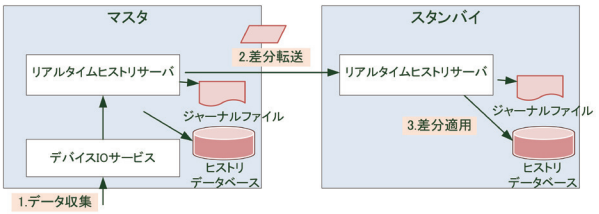

図12　データ冗長の仕組み

ePREXIONでは，マスタサーバからスタンバイサーバに転送する情報量が少なくなるように最適化を実現しており，マスタサーバとスタンバイサーバの通信帯域が比較的狭い場合でも使用できるよう設計している。

リアルタイムヒストリサーバは，収集したデータの差分をジャーナルファイルというファイルを用いて管理する。ジャーナルファイルはスタンバイサーバをメンテナンスなどでシャットダウンしている場合に，マスタサーバからスタンバイサーバに転送できなかった差分情報を保持する。メンテナンス中に転送されなかったデータは，スタンバイサーバのメンテナンスが終了した後に，マスタサーバのジャーナルファイルの情報をスタンバイサーバに転送することによって再同期をすることが可能となる。

3.2　データ収集を行う経路を冗長する経路冗長機能

ゲートウェイを2系統用意している制御システムからデータ収集を行う場合は，2系統両方からデータを収集することにより可用性を高められる。またデータ収集する設備までのネットワーク経路が冗長化されている場合にも対応することができる。これらの機能を経路冗長機能と呼んでいる。

冗長システムの構築においては，いくつかの課題がある。冗長機能に直接かかわる他系統のゲートウェイの監視方法や異常発生時の検出とスピーディな系統の切替えなどの課題もあるが，今回の報告では3つの課題と解決方法について紹介する。

・二重化冗長と待機冗長それぞれの課題
・コンフィギュレーション情報一元化の課題
・メンテナンス操作の一元化の課題

3.2.1　冗長方式を混在できる仕組み

冗長方式には二重化冗長と待機冗長がある。二重化冗長方式では，2系統から収集を行うために，ゲートウェイに負荷をかけてしまうという課題があった。また，待機冗長方式では，待機状態から稼働状態に移行するまでの間の収集データが欠損してしまう。

図13に示すようにデバイスIOサービスでは，複数のコネクションごとに冗長方式の選択ができる仕組みとした。この仕組みを使うことで，冗長を行う制御システムの要件に

合わせて二重化冗長にするか待機冗長にするかもしくは冗長しないかを選択することができる。

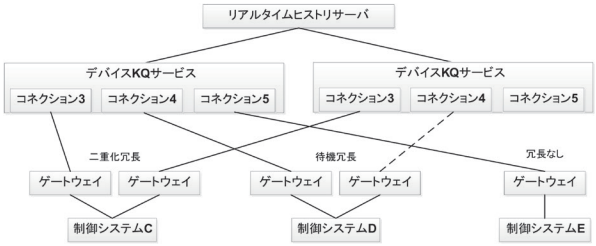

図13　経路冗長の仕組み

3.2.2　コンフィギュレーション情報を一元化する仕組み

冗長システムでは，2系統のデバイスIOサービスに同様なコンフィギュレーションを行う必要がある。しかし別々にコンフィギュレーションを行うのでは，操作に間違いが生じる恐れがある。

そのため2系統の設備の定義をセットでコンフィギュレーションすることとした。また当初は冗長で稼働していなかったシステムを冗長化することも可能である。さらに制御システムのゲートウェイそれぞれの定義が異なり，正しく収集できなくなることを防ぐために，定義が異なる場所を表示することができる。これらの仕組みにより，コンフィギュレーション情報の食い違いを防止している。

3.2.3　メンテナンス操作の一元化の仕組み

ゲートウェイとの接続や切断などのメンテナンス操作をそれぞれに行うとコンフィギュレーションと同様に間違いが生じる恐れがある。

2系統のデバイスIOサービスをペアとして扱うことを可能とした。それによりゲートウェイとの接続･切断や，状態取得を同時に2系統に行うことが可能になる。

4.　おわりに

今後IoTにより様々なセンサからデータが蓄積されるようになる。しかしながらセンサから収集したデータは，そのままで意味のあるデータとして活用することはできない。ePREXIONでは単なるデータの蓄積ではなく，演算機能により有意なデータを作り，有意なデータのみの長期蓄積が可能となる。

また機械学習などの技術を用いてビッグデータを解析するには，解析対象となるヒストリデータを保護することや確実に収集されることも重要となる。そのため，収集したデータの保護，収集経路の冗長という機能でロバスト性を向上させた。

クラウド時代の到来により，企業全体という規模から世界規模へとシステムはスケールアップしている。また機械学習などを用いたアプリケーションも，これまで以上に早いスピードでユーザーが活用できることが必要となっている。これら時代の変化に対して十分な貢献ができるように進化を続けたい。

＜著者所属＞
棚澤　武志　アズビル株式会社
ITソリューション本部ITソリューション開発部
菊地　健一　アズビル株式会社
業務システム部