Vol.2016-OS-136 No.10
2016/3/1

Plan9 によるコマンドベース MapReduce と IoT への応用

中原健志[†1]　佐藤　未来子[†1]　並木　美太郎[†2]

概要：　ネットワークを介して様々なデバイスを接続し，ホストから多数のデバイスの制御を可能とする IoT（Internet of Things）が普及しつつある．しかし，IoT ではホストから個々のデバイスを制御する場合，それぞれのデバイス独自のミドルウェアやライブラリを利用しなければならない場合が多い．このため，IoT を利用するアプリケーションはそれぞれのミドルウェアやライブラリへ対応する必要がある．そこで本研究では Plan9 のアクセス透過性を用いて，個々のデバイスをファイルに仮想化し，それぞれのファイルへの読書きにより，デバイスを統一的に制御可能な IoT 基盤の構築を検討した．また，IoT ではネットワークの能力に制限がある場合が多いが，これについてホスト側でデバイスの情報を一時的に持たせることで解決する．加えて，本 IoT 基盤を用いて収集された大量のログデータを処理に，コマンドベースの MapReduce を Iot 向けに拡張して，用いることを検討した．

キーワード：IoT，MapReduce，　Plan9，センサーデバイス，　センサーネットワーク

1. はじめに

近年，ネットワークの高性能化が進み，様々な場所でネットワークを利用可能になりつつある．一方で，組み込み機器では，高性能な製品が低価格で広く流通するようになった．これらの背景から，様々なものにセンサーデバイスなどの組み込み機器を取り付け，ネットワークへ接続することが可能になった．これにより，今まで組み込み機器をネットワークへ接続していないときはできなかった，機器の遠隔操作や，機器同士で連携した制御が可能になった．組み込み機器をネットワークへ接続し，情報の交換や制御を可能にする技術を総称して「IoT」（Internet of Things）と呼び，現在，関心が高まっている．IoT の活用例として，一人暮らしの高齢者の見守りシステム，太陽光発電の遠隔監視，倉庫の温度管理，部屋の空調管理などが知られている．

一方で，IoT が普及するためには，ソフトウェア製作者が簡単にネットワークで接続されている機器へアクセスし，制御できることが重要になる．しかし，既存の多くの IoT 向けシステムでは，接続される機器によって通信方式が異なる場合が多い．接続する機器ごとの通信方式が異なると，プログラム中で機器との通信に関する部分を，対象とする機器ごとに変更する必要がある．また，プログラム中で通信の制御を行う必要があるため，「通信待ち」や「送信」など，本来行いたい内容以外の処理も記述する必要がある．また，プログラムは組み込み機器がネットワークを介したリモート，もしくはプログラムが動作しているローカルマシンに接続されているかを区別しなければならない．

そこで，本研究では，センサーなどの組み込み機器を共通した手法で制御・データの取得を可能にする IoT 基盤の提供を目指す．また，位置透過に各機器をアクセス可能にすることで，利用者がネットワークを気にせず，機器を利用可能にする．

これらに加え，IoT では機器で取得された過去のデータをログとして保存し，内容を解析することで，機器が設置されている場所の環境改善などへ活用されることが期待されている．一例として，日々の生活データを保存・解析することで，病気の前兆などを調べるライフログが挙げられる．しかし，接続される機器が増えるにつれ，収集されるデータは膨大になるため，データの解析に時間を要する場合がある．このような，巨大なログデータの解析に，分散処理のモデルの一つである MapReduce を用いることを検討した．多くの MapReduce の実装では，行いたい処理内容をプログラムで記述する必要がある．そのプログラム作成の手間から，データの解析を簡単に行うことが難しい．そこで本研究では，IoT 基盤を用いて収集されたデータを，プログラムではなく，使い慣れたコマンドによる MapReduce を用いることで，手軽にデータの解析を実現することを目指す．

本稿では，Plan9 の位置透過性・アクセス透過性の機能を用いることで簡単に実現可能な IoT 基盤と，本 IoT 基盤にて収集したデータを解析するためのコマンドベースの MapReduce 実行基盤について述べる．以下，本研究では，組み込み機器を「ノード」，組み込み機器へ接続・制御するものを「ホスト」と呼ぶ．

2. 先行研究

本章では IoT，及び接続機器の管理手法に関する研究について説明する．

先行研究[1]では WebSocket[2]を用いて，ホストとノード間の接続，および制御を行っている．WebSocket はウェブサーバとウェブブラウザ間の通信のために開発されたプロトコルの一種である．WebSocket では，コネクション確立後の制御を，プログラムが主体となって行う必要がある．

†1 東京農工大学　工学府
Tokyo University of Agriculture and Technology
†2 東京農工大学　工学研究院
Tokyo University of Agriculture and Technology

そのため，ホスト上で動作するアプリケーションがノードに接続する場合，コネクション確立後の制御をアプリケーションが行わなければならない．もし，接続対象の機器が変わった場合には，アプリケーションのプログラム中で，コネクション確立後の制御に関する部分の改変が必要となり，アプリケーション作成者への負担が重い．

先行研究[3]では，ノードをファイルへ仮想化し，ホストマシンの名前空間へ取り込む．ノードが取得した情報については，ノードが仮想化されたファイルを読み込むことで取得可能である．また，ノードの制御については，仮想化されたファイルへ制御用の文字列を書き込むことで，その文字列の内容がノードへ送られ，実際の制御が行われる．

ファイルへ仮想化する利点として，ノードとホスト間の通信はファイルへの読み書きで実現され，利用者はプログラム中でノードとの通信に関する記述をする必要がなくなる．一方で，この研究ではネットワークの接続が一時的に切断され，その間に機器が取得した情報については考慮されていない．そのため，ノードとホスト間の接続が一時的に切断され，その間にノードが取得した情報に変化があったとしても，それらの情報を再接続後に取得することはできない．

先行研究[4]は，センサーなどから収集されたデータの解析を主目的とした，IoT 向けの MapReduce 実装の一つである．特徴として，IoT で収集されるデータは，一般的にそのセンサーが設置してある場所やデータが取得された時刻などと関連付けされることが多い．このようなデータは空間（Spatial）データと呼ばれ，データ解析時には，時間や場所といった関連付けされた情報を考慮する必要がある．そこで先行研究では，MapReduce の代表的な実装である Hadoop へ，空間データを処理するための機能を追加することで，空間データの解析を容易なものにしている．しかし，行いたい処理内容を独自の言語で記述しなければならず，使いなれているプログラムを処理へ応用することが難しい．

3. 課題

これらの課題をまとめると，以下の３点が挙げられる

- ノードの情報取得に独自の API を利用する必要がある
- ノードとホストの通信路が切断中に，ノードで取得されたデータを，再接続後にホストが取得できない
- ノードから収集された大量のデータを，MapReduce を用いて解析を行いたいが，使いなれているプログラムを応用することが難しい

4. 目標

本研究では，組み込み機器を透過的にアクセス・制御可能な IoT 基盤を提供する．本研究で提案する IoT 基盤では，ノードの制御やデータの取得において，独自の API を利用せずに共通したインターフェースを提供することで，プログラムのノードへの依存を減らす．また，ホストとノード間の通信に関する部分をプログラムから隠ぺいすることで，共通の方法でノードを利用できるようにする．例えば，ホスト上で UNIX における「cat」や「echo」コマンドを用いて，ノードを制御可能にする．

これらの目標を実現するために，本研究ではノードとホスト間の通信に Plan9[5]，および 9P[6]を用いる．Plan9 はベル研究所が開発した分散環境向けの OS（Operating System）である．Plan9 の特徴として，名前空間の編成が自由に可能である．本研究では，この機能を用いて他のマシンとの間で名前空間を共有する．一方，9P は Plan9 で用いられているネットワークファイルシステムの通信プロトコルである．9P の特徴として，様々な計算機資源をファイルへ仮想化し，通信することが可能である．これにより，様々な資源を共通した手法で利用できる「アクセス透過性」を実現している．ノード，ホストはそれぞれ 9P のプロトコルスタックを持ち，相互に通信可能であるとする．

提案する IoT 基盤では，ノードとホスト間の通信が切断された場合でも，ノード内にキャッシュを持たせることで，切断中に取得したデータを一時的に保存しておき，再接続後にホスト側から取得可能とすることを目指す．これにより，消費電力の制限が厳しいセンサーデバイスなど，必要な時のみネットワークへ接続することで省電力化を実現しているデバイスでは，ネットワーク切断中に取得したデータをデバイス側で一時的に保存しておくことでネットワークの接続回数を減らすことができる．

また本 IoT 基盤を用いて，収集されたデータの解析に MapReduce を用いる．MapReduce を用いることで，解析の処理を複数のマシンで分散させ，処理の高速化を目指す．IoT 基盤で収集されたデータを複数のマシンで共有する方法については，Plan9 の名前空間の機能を用いることにより他のマシンからも，ノードのデータを取得可能にする．

一方で，MapReduce での処理内容の記述について，プログラミング言語などの制限を減らしたい．そこでコマンドによる MapReduce を実現することで，使い慣れているプログラムを容易に IoT のデータ解析へ応用可能にする．

5. 提案手法

本研究の目標はネットワークを介して接続されている組み込み機器をプログラムから透過的にアクセス・制御可能な IoT 基盤の提供である．これを実現するために，３章で述べた課題の解決策として，接続されている組み込み機器をファイルへ仮想化する手法を提案する．機器のファイルへの仮想化は，Plan9 のアクセス透過性の機能を用いることで実現する．機器をファイルへ仮想化する利点として，ファイルの読書きはプログラムの機能の中でとても基本的なものである．機器をファイルへ仮想化することで，様々

なプログラムが機器をファイルの読書きという，とても基本的な方法で利用することが可能になる．またファイルへ仮想化することで，３章で述べた「ノード間のネットワーク」の課題については「ファイルツリー中に仮想化したファイルを配置」することで解決する. これについても, Plan9 の位置透過性の機能である，名前空間が自由に編成できる機能を用いて実現する.「通信路の遮断」の課題についても「ファイルキャッシュ」を持たせることで解決することができる．以下，３章で述べた課題に対する解決手法について説明する．

(1) IoT ノードのファイルへの仮想化

ノードの制御に独自の API を用いる手法では，ノードを利用するプログラムが API に対応する必要があった．しかし，ノード独自の API を用いるため，プログラムの汎用性が減り，異なった種類のノードでは，プログラムを再利用することができないという問題があった．

この問題の解決策として，Plan9 によりノードをファイルへ仮想化することを提案する．本仮想化によりノードとの通信はファイルへの読書きによって実現される．具体的には，ノードから情報を取得する場合，ノードを仮想化したファイルの内容を読み込むことで実現できる．また，ノードの制御は仮想化したファイルへ，行わせたい制御の命令を書き込むことで実現できる．

ノードをファイルへ仮想化する利点として,「ファイルの読書き」という極めて共通した手法を用いることで，特定の API などへの依存を減らすことができ，プログラムの汎用性が増す．また，ノードの制御や管理について，プログラムが特定の API に依存する必要がないため，日常的に用いているコマンドを応用可能である．加えて，プログラム中で通信に関する記述が必要でなくなるため，プログラム作成者は行わせたい処理内容以外の余計な記述を減らすことができる．

またネットワークの差異をユーザから隠蔽し，共通の方法でノードへアクセスできるようにしたい．これを達成する手法として，ノードをホストの名前空間へ配置することを提案する．この手法の利点として，様々なノードが接続された場合でも，ネットワークを意識せず，ホストの名前空間を辿ることでノードへのアクセスが可能である. また, 利用者の目的に合わせて，ホストの名前空間の編成し，ノードのディレクトリを再配置することで，目的に沿った名前空間の生成が可能である．これらの機能を Plan9 の位置透過性を用いることで，容易に実現する．Plan9 ではネットワークを介して，接続されているマシンの名前空間を自分のマシンの中へ取り込み，好きな場所へ自由に配置可能にすることで，リモートとローカルを区別せずに利用できる位置透過性を実現している．

(2) ファイルキャッシュのサポート

ノード・ホスト間のネットワークが一時的に切断された場合，ホスト上で動作するプログラムに対して，ネットワークの切断を隠蔽する．一方で，ネットワーク切断中にノードが取得したデータについて，再接続後に確認できるようにしたい．これらの目標に対して，ノードとホストの両方にファイルキャッシュを持たせることを提案する．

ノード側ではネットワーク切断中に取得したデータをノード内のキャッシュに保存しておき，再接続後にホストへ送る．一方，ホスト側ではネットワークが切断される直前に，ノードから受信したデータをキャッシュへ保存しておく．ネットワーク切断中に，プログラムがノードのデータを要求してきた場合には，保存されたキャッシュのデータを渡すことで，プログラムからネットワークの切断を隠ぺいすることが可能になる．また，ホストのキャッシュのデータを消さずに蓄積した場合は，ノードが取得した過去のデータを参照できる利点がある．ただし，ノードの取得するデータの種類によっては，リアルタイム性が求められるものもあるため，ホスト側のキャッシュについては，利用の可否を設定可能にする．

(3) コマンドベースの MapReduce

ノードが収集したデータの解析に MapReduce を用いることで処理を複数のマシンで分散化し, 高速化を行いたい. しかし，MapReduce の多くの実装では，処理内容の記述に独自の言語を使う必要があるため，手軽に処理を行うことが難しい．そこで，UNIX のパイプとフィルタの考え方を基に，データの分割・統合の機能を追加することで，コマンドによる MapReduce を実現する．利点として，日頃使い慣れているプログラムを簡単に，MapReduce へ応用可能である．

また，マシン間のデータの共有については，Plan9 の位置透過性の機能を用いて，リモートマシンの名前空間を自分の利用しているマシンの名前空間と共有させることで実現する．これにより，リモートマシン上で動作するコマンドは，自分が利用しているマシンの名前空間中にあるファイルへ簡単にアクセスすることができる．

また，IoT では，データ本体と取得された時刻や場所などの関連データが紐付られて処理されることが多い．これらの関連付けを名前空間のディレクトリ名を基に処理を行うことで実現する．

6. IoT基盤の設計

6.1 システム構成

本システムは３つのサービスとして「ノード」,「IoT 基盤（ホスト）,「ログ情報保存サーバ」から構成される．それぞれはネットワークを介して 9P で接続されている．（図１）以下，それぞれのサービスの役割を説明する．

- 「ホスト」は複数のノードからの接続を受け，ノードの管理・制御を行う．
- 「ノード」は組み込み機器であり，設置されている場所で計測される情報を提供する．

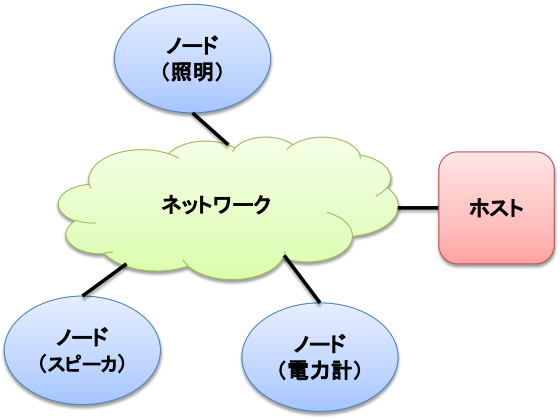

図 1　システム概念図

以下，第５章で述べた提案手法を本システムへ適用するための設計について述べる．

6.2　ノードの機能

6.2.1 ノードが提供する機能

ホストでノードへ接続した際に，ノードの情報を UNIX での「cat」コマンドで表示させ，制御については「echo」コマンドでファイルへ書き込むことで実現したい．そのため，ノードは下記の機能をホストへ提供する．

- ノードで取得したデータをファイルへ仮想化し，ファイル入出力の出力を通して提供する．
- ノードの制御インタフェースを，ファイル入出力の入力を用いて提供する．

ノードはこれらのインタフェースをファイルツリーの形で，ホストへ公開する．ホストはノードのファイルツリーを，Plan9 の位置透過性の機能を用いて，自身の名前空間へ取り込むことで，ノードへのアクセスを可能にする．

6.2.2　名前空間へのノードの取り込み

ノードをホストへ接続させるとき，ノードの IP アドレスとポート番号を指定してマウントすることで，ノードが提供するファイルツリーがホストの名前空間へ取り込まれる．取り込み方として，ホストではノードの種類や場所などに応じて，ノードのファイルツリーをホストの指定された名前空間下へ取り込む．　Plan9 では名前空間を自由に編成することができる．名前空間の取り込みについては，Plan9 の位置透過性の機能を用いて，実現している．ノードの名前空間の取り込み先は，ユーザの設定が可能である．例として，図２は，「sensor/」というディレクトリの下に「light」と「audio」という２つのディレクトリがあり，前者に照明，後者にオーディオ機器のノードを集めたいとする．この場合，接続されたノードの種類が照明であった場合は，「light」のディレクトリ下へ，オーディオ機器であった場合は「audio」ディレクトリ下へ，ノードの名前空間を自動的に取り込む．ノードの種類の判定については，ノードの名前空間下にある「property」ファイルの「type」の項目を用いて判別する．もし，ノードの「property」ファイルにノードの種類が書いてなかった場合，ユーザで接続したノードの種類と取り込み先のディレクトリの対応付けを設定することができる．

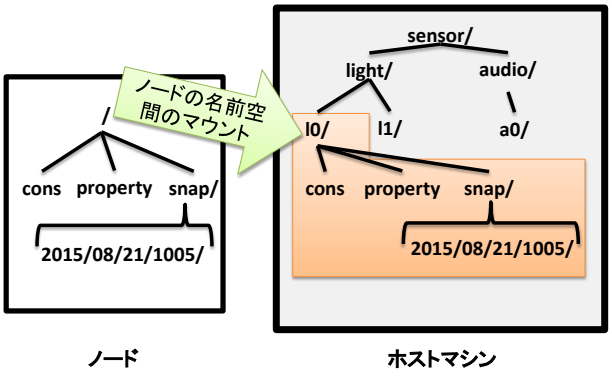

図 2　ノードからホストへの名前空間の取り込み

6.2.3 ノードが提供するファイルツリー

ノードは自身をルート「/」とするファイルツリーを持つ．ファイルツリーには幾つかのファイル・ディレクトリが存在する．これらはノードの機能，情報をファイルへ仮想化したものである．本システムで，ノードが必ず保持しなければならない，ファイル・ディレクトリを図３に示し，各項目について説明する．

/ctl	ノード制御インターフェース
/property	ノードの識別 ID，設定を含む
snap/	キャッシュデータの保存

図 3　ノードが必ず保持するファイル・ディレクトリ

- **ファイル：/ctl**

ファイル「ctl」は 9P のアクセス透過性の機能を用いて，ノードの制御インターフェースを仮想化したものである．ユーザは ctl へ文字列を下記込むことで，ノードの制御ができる．また，ノードの情報を取得したい場合は，ctl を読み込むことで可能である．

- **ファイル：/property**

ノードがホストへ接続され，Plan9 の位置透過性の機能を用いて，ノードの名前空間がホストの名前空間へ取り込まれる際に，ノードとホストのディレクトリの対応付けを行う必要がある．しかし，種類が同じノードが複数ある場合，名前空間の構成のみでノードを識別することが難しい．そこで，ノードを識別するために，それぞれのノードに対して，あらかじめ固有の ID を割り振る．この値を含んでいるのが property である．その他にキャッシュの有無などのノードを管理する際に必要な情報・設定が含まれる．
図４に，property の内容例を示す．

```
id:201510
type: light
cache: on
```

図 4　property の内容例

property では設定項目は「設定 : 値」の形式で記述される．表２の例では，項目「id」がノードに割り振られた固有の ID，「type」がノードの種類，「cache」がキャッシュの利用の可否を意味する．

- **ディレクトリ：snap/**

ノードがネットワークから切断されている間に，ノードが新しいデータを取得した場合，ネットワーク再接続後にホストはノードでのデータの変化を取得することができない．そこで，ネットワーク切断中に，新しいデータを取得した場合，取得した内容をノード内に一時的にキャッシュとして保存する．データの保存は，ノードの「snap/」ディレクトリ以下に，情報が取得された時間を基に「/year/month/day/hour」でディレクトリが生成され，制御インターフェースである「ctl」ファイルの内容を読みだしたものが，先ほど生成したディレクトリ下にファイルとして保存される．

ホストとの接続が再確立した場合は，Plan9 の位置透過性の機能を用いて，ホスト上の名前空間で自身のノードを示すディレクトリ下の「snap/」ディレクトリへ，ノードの「snap/」ディレクトリを重ね合わせる．そして，ノードでキャッシュされたデータを，ホストの「snap/」ディレクトリへコピーする．これにより，ホスト上のアプリケーションはネットワーク切断時のノードの情報を取得することが可能になる．

6.3 ノード・ホスト間でのネットワーク切断への対応

ノードとホスト間で一時的にネットワークが切断された場合，ホストの名前空間から切断されたノードのファイルツリーを消さずに残す．これにより，ノードが再接続された場合は前回接続時に配置されたファイルツリーへノードのファイルツリーが再配置され，ネットワーク切断ごとに，ノードのファイルツリーの位置が変更されることを防ぐ．

これを実現するには，接続されていたノードのファイルツリーが，ホストの名前空間中で，どこに対応していたかを判断する必要がある．これを判断するために，ホストはノードに割り振られた固有の ID をノードの「property」ファイルから読み出し，ホストのディレクトリとの対応を関連付けるテーブルを作成する．再接続時には，ホストが持つテーブルとノードの固有 ID を調べることで，ホストの名前空間でノードのファイルツリーが移動することを防ぐ．

6.3.1 キャッシュの構成

本システムでは，ノード・ホストの両方がキャッシュを有する．これはノードとの接続が一時的に切れた場合に，ホスト側ではノードとの切断をプログラムから隠ぺいし，透過性を保つために用いられる．

ノード側では，ホストから切断されている間に取得したノードのデータを保存し，ホストと再接続した際に過去のデータを参照できるようにするために用いられる．ホスト，ノードのキャッシュの設計について，下記に述べる．

6.3.2 ホスト側のキャッシュの設計

ノードが接続された際に，ノードが提供する「snap/」ディレクトリをホストへコピーする．そして，ノードの「ctl」ファイルへのアクセスを監視し，ホスト上のプログラムがノードのデータを読み込んだ場合，内容をホストの「snap/」ディレクトリの以下へコピーする．また，一定期間以上が過ぎたデータに関しては削除される．

一時的にノードとの接続が切れた場合でも，アプリケーションからノードのデータ取得を透過に行いたい．そのため，ノード切断時にホストのプログラムが「ctl」ファイルへアクセスした場合，「snap/」ディレクトリで保存されているデータの中で最も新しいものを返す（図３）より，ホスト上のプログラムはノードとのネットワーク接続状況を気にせず，「ctl」ファイルを透過に利用可能である．ただし，ノードの種類によっては，キャッシュの利用が相応しくない場合がある．このため，「ctl」ファイルのキャッシュについて，利用の可否をノードの「property」ファイルでで設定する．

6.3.3 ノード側のキャッシュの設計

ノードがネットワークから切断されている間に新しいデータを取得した場合，ネットワークが再接続しても，ホストはノードでのデータの変化を知ることができない．そこで，ネットワークが切断されている間に，新しいデータを取得した場合，それらの情報をノード内に一時的にキャッシュとして保存する．データの保存は，「snap/」ディレ

クトリ以下に情報が取得された時間で「/year/month/day/hour」のディレクトリが生成され，ノードの「ctl」内容を読みだしたものがファイルへ仮想化され，保存される（図４）

ホストとの接続が再確立した場合は，キャッシュされた情報をホスト上の名前空間下にある「snap/」ディレクトリへ生成したディレクトリ・ファイルをコピーする．これにより，ホスト上のアプリケーションはネットワーク切断時のノードの情報を取得することが可能になる．

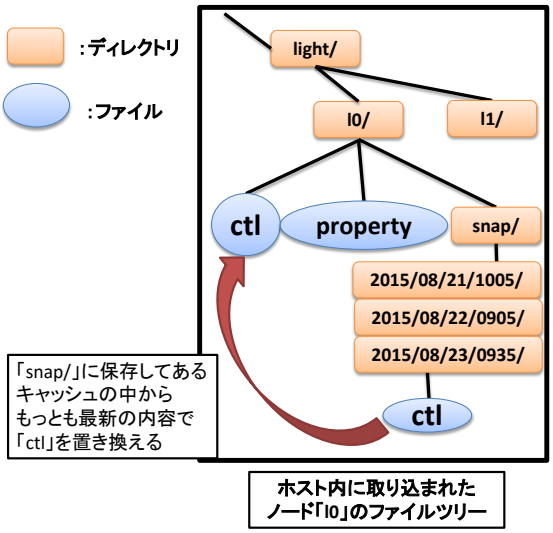

図 5　ホスト側でのキャッシュによる
ネットワーク切断の隠蔽

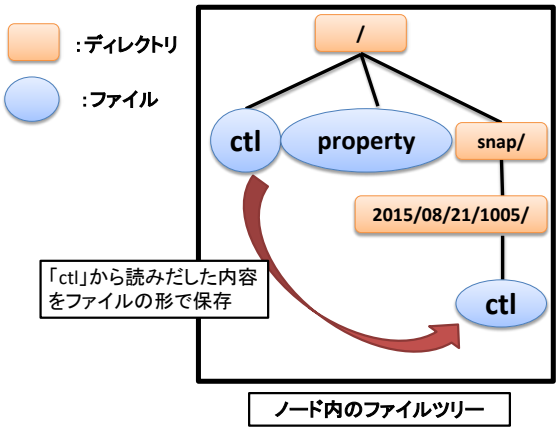

図 6　ノードでのキャッシュファイル生成

6.3.4 キャッシュを応用した，ノードの過去データ保存

ホスト側のキャッシュを削除せず保存することで，ノードの過去のデータを蓄えることができる．過去のデータを蓄える利点として，蓄えたデータを解析し，特徴や傾向などを調べることで，ノードが設置してある場所などの改善に繋がる可能性がある．ノードはファイルへ仮想化されているので，ファイルサーバのバックアップの考え方を応用し，指定されたノードのファイル内容を読み出し，スナップショット形式で保存する．保存先については，ノードごとに分けてデータを保存するために，それぞれのノードに割り振られた固有 ID に対応するディレクトリに，情報取得時刻に基づき「/year/month/day/hour」の名前のディレクトリを作成し，そのディレクトリ下へ保存する．ノードの過去データについては，他のホストでも利用されるため，他のホストからも共有可能にする．

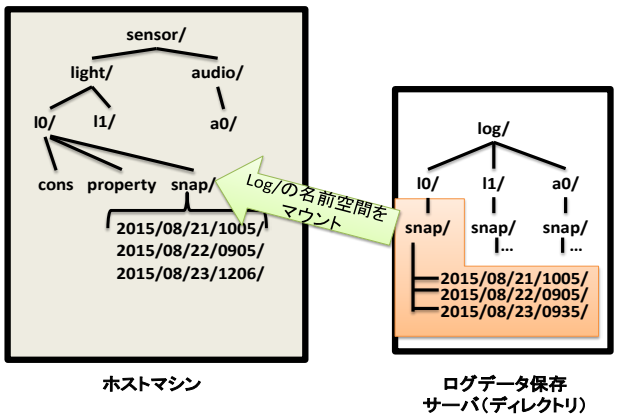

図 7　ログ情報のマウント

7. MapReduceによるノード情報の分析

7.1　ノードの収集データの活用

IoT の活用例として，過去のログデータを解析し，有益な情報を求めるものがある．代表的なものに，ライフログが挙げられる．ライフログでは，日々の生活状況をセンサーなどで計測し，集めたデータを解析することで生活習慣の傾向を調べ，改善などに役立てるものである．またエアコンなどの電力と部屋の温度をセンサーで計測し，温度と電力の相関を調べることで，効率的なエアコンの制御を行う事例がある．

これらの事例から，ノードの過去データの解析を行いたい場面がある．しかし，ノードの数が増えるにつれ，収集されたデータ量は膨大になり，単体のマシンでは処理が難しい場合が考えられる．そこで，収集された過去のデータの解析に分散処理を用いたい．

7.2　コマンドによるMapReduce

分散処理のモデルとしては，MapReduce[7]が広く知られている．MapReduce では入力データを複数に分割する Map 処理，及び分割されたデータをネットワークで接続されている複数のマシンで処理し，それらの結果をまとめる Reduce 処理の２つから成り立つ．MapReduce の実装として Hadoop[8]や Apache Spark[9]などが有名である．しかし，こ

れらの実装では処理対象のプログラムを MapReduce 用に書き換える必要があり，既存のプログラムをそのまま利用することが難しい．そこで，UNIX の「パイプとフィルタ」の考え方を基に，シェルに Map 処理である「データの分割」，および Reduce 処理である「データの結合」を導入する．そして，ユーザが指定したコマンドをネットワークで接続されている複数のリモートマシンで実行し，それぞれのマシンへ分割したデータを渡し，それらの処理結果を結合することで既存のコマンドを変更せずに利用可能な MapReduce の作成した[10]．この，コマンドベースの MapReduce をノードで取得した過去データの解析や整形，及びノードの制御へ応用することを検討している．

本 MapReduce の特徴として，Plan9 の位置透過性の機能を用いることで，複数のマシン間での名前空間の共有が挙げられる．この機能を用いることで，ホストで接続しているノードの名前空間を，ほかのマシンが参照し情報を取得することができる．ノードの情報取得はファイルを読み込むことでできるため，通常のコマンドを処理に使うことができる．また，ノードの過去のデータの解析についても，ファイルの形で保存されるため，ノードから取得したデータの処理に用いたプログラムを容易に応用することが可能である．

しかし，課題として，本 MapReduce では一つの大きな入力データを分散させて処理することを対象としている．しかし，ノードから収集されるデータは多数の小さなファイルから構成されるため，そのままでは用いることができない．また，IoT のデータは時系列と密接に関係しており，処理結果の出力については時系列が分かる形で行いたい．そこで，本 MapReduce を IoT の処理へ応用するため，次の機能拡張を行う．

(1) **複数のファイルからのデータ入力への対応**
(2) **時系列を保った処理データの出力**

7.3 Plan9 によるコマンドベースMapReduceの設計

7.3.1 複数のファイルからのデータ入力への対応

本 IoT 基盤により保存されるノードの過去のログデータは，取得された時間ごとにディレクトリへ分けられ，ファイルの形式で保存される．そのため，コマンドベースの MapReduce で処理を行わせたい場合，処理対象となる入力データは複数のファイルとなる．そのため，分散シェルを複数のディレクトリ・ファイルの入力に対応させる．そこで，分散シェルへ図８の記号を追加・拡張する．

```
<//{入力ファイル，条件}
```

図 8　複数ファイル入力記号

複数ファイル入力記号では，記号で囲まれた条件のファイルが，分割されずに並列処理記号内のコマンドへ渡される．入力ファイルの指定方法として，分散シェル上でディレクトリ・ファイルのパスを直接記入する，もしくは正規表現を用いて入力対象のディレクトリ・ファイル名の条件を定義し，条件に合致したファイルが入力対象にする．入力対象になったファイルは，シェルが自動的に複数のリモートマシン上で動くコマンドの標準入力に設定される．

7.3.2 時系列を保った処理データの出力

時系列について，本 IoT 基盤では，ノードの過去データが保存されているディレクトリの名前が，データの時系列を表している．そのため，MapReduce 時に入力元のディレクトリの名前を基に，出力時のファイル名を決め，すべてのログデータの処理が完了した時点で，ファイル名でソートを行うことで時系列を保ったデータ出力が可能である．

7.3.3 本MapReduceを用いた空調機器の制御例

本 MapReduce を用いた例として，ノードとして空調機器が繋がれている状況で，ノードへ空調機器の制御情報を書きこむことで制御する例を示す．空調機器には温度センサーがついており，ノードが提供する情報として，空調機器の設定設定（SetTemp:），運転モード（Mode:），室温（RoomTemp）がわかるものとする．また，ノードの制御については，「ctl」ファイルへ「sw :0」（電源切），「sw: 1」（電源入）を書き込むことで電源の入切が可能である．ここで，接続されているすべての空調機器の中で室温が２５度以下の場合，その空調機器の電源を切りたいとする．その場合，図９のコマンドをシェルへ入力することで，実現できる．

```
//{grep –o 'RoomTemp:' | grep –o [0-9]* | swTemp
25 }// <//{sensor/AC/*/ctl} >//{sensor/AC/*/ctl}
```

図 9　コマンドベースの MapReduce による，空調機器の制御の例

上記の記述内容では初めに「sensor/AC/*/ctl」から空調機器の情報を読み込む．読み込まれた情報の中から「grep」コマンドを用いて，室温を示す「RoomTemp」が含まれる行を探し，室温の値を出力する．出力された値は「swTemp」へ渡される．「swTemp」は入力された値が引数よりも小さい場合は「sw: 0」，大きい場合は「sw: 1」を出力するコマンドとする．「swTemp」が出力した結果を，入力元のディレクトリ内にある「ctl」ファイルへ書き込むことで，空調の制御が行われる．

8. 現状

現在，IoT 基盤およびコマンドベースの MapReduce の実

装を進めている．IoT 基盤に関しては，Raspberry Pi2 へ USB 経由で接続しているセンサーデバイス（Phidgets InterfaceKit 8/8/8 1018）を名前空間へマウントするためのドライバーの作成を行っている．Raspberry Pi2 上では，Plan9 が動作している．今回の実装では，Raspberry Pi2 をセンサーノードとして使う．現在，実装中のセンサーノードが提供する名前空間を図６に示す．

Phidgets InterfaceKit 8/8/8 はセンサーの接続ポートを８つの持つセンサーデバイスである．ポートへ接続されるセンサーはアナログセンサーであり，本センサーデバイスで，接続されたセンサーの抵抗値の変化を A/D コンバータによりデジタル値へ変換する．そのため，ルートディレクトリ（/）以下にはファイル「ctl」（Phidgets InterfaceKit 8/8/8 本体の制御），「property」（Phidgets InterfaceKit 8/8/8 本体の管理情報を提供）が存在する．ルートディレクトリ（/）直下には，ディレクトリ「phidgets」が存在，その下のディレクトリには「0」～「7」のディレクトリが存在する．これは，Phidgets InterfaceKit 8/8/8 がセンサーを接続するためのハブの役割をしており，接続可能なセンサーの数が８のためである．「0」～「7」のディレクトリは接続されるセンサーを仮想化しており，それぞれのディレクトリにはファイル「ctl」（接続されたセンサーの情報を取得），「property」（接続されたセンサーの管理情報を提供），及びディレクトリ「snap/」（接続されたセンサーのキャッシュ）が存在する．

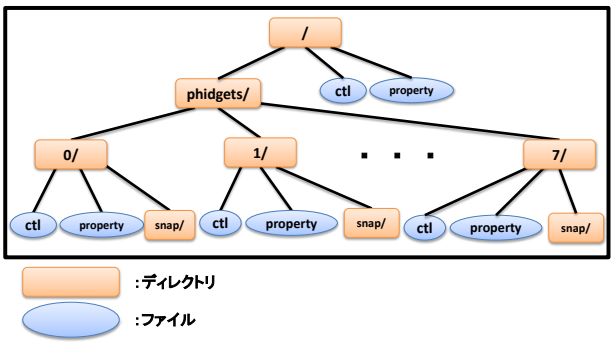

図 10　センサーノード「Phidgets InterfaceKit 8/8/8」が提供する名前空間

9. おわりに

本稿では Plan9 のアクセス透過性を用いて，個々のデバイスをファイルに仮想化し，それぞれのファイルへの読書きにより，デバイスを統一的に制御可能な IoT 基盤について提案した．ノードの管理手法の検討を行った．提案手法の特徴として，ノードのファイルへの仮想化，名前空間とファイルキャッシュによる透過性の確保，そしてノードの過去のデータの保持などが挙げられる．

一方，課題として，ノードはキャッシュ保存の際に情報取得時刻に基づいて，ディレクトリを作成し，データを保存する．そして，ノードがホストへ接続した際に，取得したデータをホストへコピーするが，ノードとホスト間の時刻設定に大きな差があった場合，データの時間系列で不整合が生じる可能性がある．これらの課題について，今後詳細な検討を行う．また，提案手法の実装，及び評価を行う．

参考文献

[1] 清水 智可良，沼尾 雅之:RSS を用いたセンサーデータマッシュアップのためのウェブアーキテクチャ，マルチメディア，分散，協調とモバイル(DICOMO2013)シンポジウム，4H-1,2013.
[2] RFC 6455, The WebSocket Protocol
[3] Ballesteros, F.J.; Guardiola, G.; Soriano, E.; Leal, K., "Traditional Systems Can Work Well for Pervasive Applications. A Case Study: Plan 9 from Bell Labs Becomes Ubiquitous," in Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on , vol., no., pp.295-299, 8-12 March 2005
[4] Ahmed Eldawy and Mohamed F. Mokbel. "SpatialHadoop: A MapReduce Framework for Spatial Data". In Proceedings of the IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April, 2015
[5] Rob Pike and Dave Presotto and Ken Thompson and Howard Trickey, Plan 9 from Bell Labs,In Proceedings of the Summer 1990 UKUUG Conference1—9(1990)
[6] Pike, Rob; Ritchie, Dennis M., "The Styx® architecture for distributed systems," in Bell Labs Technical Journal , vol.4, no.2, pp.146-152, April-June 1999
[7] Jeffrey Dean and Sanjay Ghemawat. “MapReduce: simplified data processing on large clusters.” In Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6 (OSDI'04), Vol. 6. USENIX Association, Berkeley, CA, USA, 10-10., 2004
[8] Apache Hadoop http://hadoop.apache.org/
[9] Apache Spark https://spark.apache.org/
[10] 中原 健志，佐藤 未来子，並木 美太郎: Plan9 による分散シェルシステム，情報処理学会 「システムソフトウェアとオペレーティング・システム」，第 134 回研究発表会(SWoPP2015), Vol.2015-OS-134, No.21, pp.1-8 (2015-08-05).