Vol.2017-OS-140 No.16
2017/5/17

ユニケージ開発手法に基づく Unix ファイルシステムとシェルを用いたデータベースの構築と操作

中村 和敬[1,a)]　當仲 寛哲[1,b)]

概要：ユニケージ開発手法は UNIX 哲学に基づいたシステム開発手法であり、企業システム開発に 20 年の実績がある。近年のシステム開発では、データベースの構築に多くの場合で RDBMS が用いられる。これに対してユニケージ開発手法では Unix の機能のみを用いてデータベースを構築する。ファイルシステムにテキストファイルとしてデータを格納し、シェルを通じてコマンドでデータを処理する事により、データベースの機能を実現する。本稿では、ユニケージ開発手法に基づく、Unix ファイルシステムとシェルを用いた、データベース構築と操作の方法について概説し、その発展について述べる。

Unicage : Database development by Unix File System and Schellscript

Nakamura Kazutaka[1,a)]　Tounaka Nobuaki[1,b)]

Abstract: Unicage is a system development method based on UNIX philosophy and have been doing business in Business System Integrator area for 20 years. In recent system development, RDBMS is used in many cases for database construction. On the other hand, in the Unicage development method, a database is constructed using only Unix functions. Data is stored as a text file in the file system and processed with the command through the shell. Then the function of the database is realized. In this paper, Authors outline how to construct and operate database on Unix file system and shellscript by Unicage, and describe its development.

概要

1. Unix 哲学とユニケージ開発手法

1969 年に開発された Unix は紆余曲折を経ながらも普及を続けてきた。2001 年には Mac OSX の基盤に採用された。さらに 2016 年には Windows 10 でも Linux サブシステムという形で取り入れられ、ますます普及している。

UNIX は初期からシステム開発に関する経験に基づいた規範を蓄積してきており、これは現在 UNIX 哲学と呼ばれる [1]。UNIX 標準のユーザインターフェースであるシェル [2] は、UNIX 哲学に基づいたプログラミングが出来るようになっている。まず、UNIX のシェルコマンドは、標準入力からデータを読み込み、何らかの処理をして、標準出力に結果を出力するという、フィルタとして作成されている。また、UNIX にはプロセスの標準出力と別のプロセスの標準入力を接続してデータが流れるようにする、パイプというシステムコールが用意されており、シェルから利用出来るようになっている。これを利用して、シェルをつうじてコマンドを次々と接続してパイプラインを構築するプログラミングのスタイル (以下パイププログラミング) は、UNIX を利用する多くの人々から支持を得ている。

しかし、このパイププログラミングは、これまではもっぱら日常の簡単なデータ処理を行うために利用され、企業システムのような大規模なシステムを開発するためには使われてこなかった。

[1] ユニバーサル・シェル・プログラミング研究所
USP, NishiShimbashi 3–4–2, Minato–ku, Tokyo–to 105–0003, Japan
[a)] k-nakamura@usp-lab.com
[b)] tounaka@usp-lab.com

1.1 ファルマにおけるシステム開発

一方で、1970 年代に創業した関西地方を拠点とする薬局のボランタリチェーン、ファルマでは、同社の創業者の一人、松田康之氏が独特のシステム開発手法を編み出していた [3], [4] *。彼は NTT などの大手システム企業が中心となっていた当時のシステム化の流れに対して、彼は薬局営業の現場で業務が携わる人々が、自分自身で業務改善のための情報を得ることが出来るような仕組み、ユーザ中心のシステムを構築することに腐心していた。そこから導き出されたのが、データを中心におき、データを処理するプログラムはスクリプト言語を用いて開発し、必要に応じて書き捨てるという、当時としては画期的な考え方であった。

1.2 ユニケージ開発手法

UNIX 哲学と松田氏の思想に影響を受け、企業システム内製化のために開発されたのが、有限会社ユニバーサル・シェル・プログラミング研究所 (USP 研究所) の提唱するユニケージ開発手法 [5] である。

ユニケージでは、Unix 系 OS の機能を用いてシステムを開発する。データはファイルシステムに格納し、シェルスクリプトによりデータ処理を行なう。ユニケージは、データの管理の手法、シェルスクリプトによる開発の際の規範、実際の業務に対応するプロジェクト運営とシステム設計規範などからなる。伝票などの文字データ、数値データを処理するシステムの開発のための手法であり、業務システムの内製化に 20 年の実績がある。

近年の業務システムは関係データベースを基盤とし、それを Java などのプログラム言語を通じて操作し、Web 技術などによりユーザインターフェースを作成するという手法が一般的である。ユニケージでは、ユーザインターフェース部分に Web 技術を用いるが、それ以外の部分については、Unix のファイルシステムと、シェルスクリプトによって開発を行なう。独自のシェルコマンドセット、 Tukubai コマンドを用いる事で、RDBMS 相当の処理をシェルスクリプトのみで実装することが可能であり、さらにデータの加工、ユーザインターフェースのための CGI スクリプトなどもシェルスクリプトで一環して実装する。これらにより、ユニケージ開発手法にはいくつかの利点がある。

デプロイ/移植が容易

ユニケージに基づき開発されたシステムはデプロイや移植が容易である。UNIX 哲学は移植性を重視すべしという方針が貫かれており、ユニケージもまたその性質を受け継いでいる。

ユニケージではデータやスクリプトは全て UTF-8 でエンコードされたテキストファイルに保存される。また、シェルスクリプトについても、基本的に bash の一部の機能のみを使用して記述する事としている。

現在の Unix 系 OS の大半で UTF-8 、および bash を使用することができるので、ユニケージに基づき開発されたシステムを特定の環境にデプロイしたり、既に動作しておりデータが蓄積されたシステムを新しい環境に移植する際にも、単純にファイルをコピーし、 Tukubai コマンドへのパスを通すだけでよい。また仮に UTF-8 、 bash を使用できない場合でも、文字コードの変換は既にさまざまなプログラムが存在し、また利用している bash の機能は限られているため、移植は可能である。

性能が高い

シェルスクリプトを用いたデータ処理は、性能が高くスケールアウトもしやすい。

シェルはそれ自体は性能の低い処理系である。しかし UNIX 哲学では、シェルスクリプトを通じて単機能のコマンドをパイプを用いて結合し、データを処理するよう示唆している。このため、 UNIX 哲学に基づいて記述されたシェルスクリプトでは、実際のデータ処理を行なうのは高度に最適化されたコマンドと Unix であり、シェルはデータ処理には介在しない。加えて、パイプを用いたシェルスクリプトは、本来的にマルチスレッドプログラムであり、現在主流のマルチコアプロセッサを有効に活用することが可能である。UNIX 哲学に基づいたユニケージでもまた同様の利点がある。

また、 RDBMS ではインデキシングなどに相当する高速化のための工夫がある。ユニケージにおけるこういった工夫は、データを複数のファイルに分割し、ファイル名により取り出せるようにするといった方法である。これは現実のファイルとのアナロジーにより、より多くの人々が直感的に理解し実施する事が可能である。

こういった理由により、ユニケージにより構築されたシステムは、容易に高い性能を発揮することが出来る。

スケールアウトしやすい

また、現在の Unix には、ssh コマンドなど、ネットワークを経由して機能を利用するための仕組みが充実している。ユニケージの独自のシステム設計規範に基づいて設計されたシステムは、こういった仕組みを利用する事で容易にスケールアウトする事が可能である。

1.3 本稿の内容

本稿ではまず、ユニケージ開発手法に基づく、Unix ファイルシステムとシェルを用いた、データベース構築と操作の方法について概説する。ユニケージは、データの管理の手法、シェルスクリプトによる開発の際の規範、実際の業務に対応するプロジェクト運営とシステム設計規範など、さまざまな要素からなるが、本稿では特にデータベースの構築と操作という側面に焦点を当てて概説する。続いて、

*実際には、松田氏と同社に勤務した多くの技術者らが共同で編み出したもののようである

ユニケージと MySQL とで同等のデータベースを構築し、その性能を比較する。最後に今後の発展について述べる。

なお Tukubai コマンドには、商用版の usp Tukubai コマンドと、オープンソースの Open usp Tukubai とがある。商用版はオープンソース版にくらべ高速化と、さまざまな細かい機能の追加が行なわれている。本稿では商用版の usp Tukubai を説明、性能評価に用いる。

2. ユニケージによるデータベースの構築と操作

本節ではユニケージによるデータベース構築と操作方法を述べる。ユニケージのカバーする領域は多岐にわたるが、今回は一つのデータベースシステムを構築するための規範に絞って概要を述べる。

2.1 ファイル形式

ユニケージでは全てのデータをテキストファイルに保存する。データは表形式であり、文字コードは UTF-8 であり、レコード区切りは LF 文字 (0x0A)、フィールド区切りはスペース文字 (0x20) である。数値データなどであっても、文字列として保存する。

例えば、売上データであれば以下のような形式で保存される。

```
店舗 ID 商品 ID 売上日 売上金額 売上個数
001 1234567890123 20170201 2280 19
001 1234567890123 20170202 2040 17
001 1234567890123 20170203 2760 23
001 1234567890123 20170204 2400 20
001 1234567890123 20170205 3000 25
```

第一レコードには、それぞれのフィールドの名前が格納される。

ユニケージではこういった形式のファイルをタグ形式と呼ぶ*。

2.2 5 段階のデータの整理

データベースは、入力データを加工して蓄積し、要求に応じて蓄積されたデータを加工して出力するものである。ユニケージにはデータベースシステムを構築するための独自の設計規範がある。これは、5 段階の枠組みに則ってデータを整理するというものである。5 段階の枠組みとは以下のようなものである。

- レベル 1: 入力データ (非タグ形式可)
- レベル 2: 確定データ
- レベル 3: 整理データ
- レベル 4: アプリケーションデータ
- レベル 5: 出力データ (非タグ形式可)

これらのデータは基本的にサービスアウト時間帯などにまとめて作成する。以下ではそれぞれについて簡単に説明する。

2.2.1 レベル 1、レベル 5

レベル 1 は入力データそのもの、レベル 5 は出力データそのものである。ユニケージのデータの保存形式は前述のタグ形式であるが、入力データ、および出力データは必ずしもテキスト形式ではない。入力データは CSV 形式のようなテキスト形式のものから、Web インターフェースを通じた入力、 Excel ファイルや対向システムからのダンプデータなど多岐にわたる。出力データも同様である。

入力データについては、入力のあった単位で、そのままファイルに保存し、バックアップを取る。これ以上の段階のデータは全て入力データが加工されて出来たものであるので、入力データの保存が完全であれば全てのデータを再現することが可能であるからである。

出力データについては、監査等の必要に応じて保存する。

2.2.2 レベル 2

前述のとおり、レベル 1 はタグ形式ではない。また、数字の文字列が入るべきフィールドにアルファベットの文字列が入っているなど、正しい形式ではないデータであることもある。レベル 2 は、レベル 1 に対して、こういったデータの変換と、正規化を施したものである。

レベル 2 を作成する際には、レベル 1 を可能な限り全て変換して作成なければならない。この段階ではどのデータがシステムに必要かどうかという判断はしない。また、レベル 2 ファイルはレベル 1 をある程度まとめて作成してよい。例えば、 Web インターフェースからの一つ一つの入力は、それぞれ別々のレベル 1 ファイルに格納されるが、それらから作成されたレベル 2 はまとめて一つのファイルに格納してよい。

2.2.3 レベル 3

レベル 2 はレベル 1 をそのまま変換したデータである。そのままでは、後でデータを取り出して加工するにはふさわしくないことも多い。レベル 3 はレベル 2 を後で使いやすいように整理したものである。詳しくは後述する。

2.2.4 レベル 4

レベル 5 は出力データであり、レベル 3 を加工して作成する。インタラクティブなシステムなど、応答時間が重要となるシステムの場合、レベル 3 を加工してレベル 5 を加工していては間に合わないことがある。レベル 4 はそういった場合に、レベル 3 をあらかじめ加工して作成しておくものである。詳しくは後述する。

レベル 4 は必ずしも作成しなければならないわけではない。必要な時に作成するものである。

2.3 レベル 3: 整理データ

前述のとおり、レベル 5 はレベル 3 から作成する。レベル 3 はマスタデータとトランザクションデータに分類され

*他にもいくつかの形式があるが、今回は取り上げない。

る。以下ではまず、それぞれどのように整理するか、またそれぞれのレコードをどのように編集するか述べる。

2.3.1 マスタデータの整理

マスタデータはキーと値の形式で保存する。この時、レコードはキーに基づいてソートしておく。キーに複数の種類の値が紐づくケースでも、それぞれの値について別々のファイルに保存する。

例えば、商品 ID に、原価、売価、商品名が紐づいている場合は、3 つのファイルを作成する。

一つ目は、商品 ID と原価の対応表であり、以下のようなファイルである。

```
$ cat SHOHINID_GENKA
商品 ID 原価
1234567890123 100
2345678901234 80
3456789012345 120
4567890123456 30
5678901234567 10
```

二つ目は、商品 ID と売価の対応表であり、以下のようなファイルである。

```
$ cat SHOHINID_BAIKA
商品 ID 売価
1234567890123 120
2345678901234 100
3456789012345 150
4567890123456 50
5678901234567 30
```

三つ目は、商品 ID と商品名の対応表であり、以下のようなファイルである。

```
$ cat SHOHINID_SHOHINMEI
商品 ID 商品名
1234567890123 商品 A
2345678901234 商品 B
3456789012345 商品 C
4567890123456 商品 D
5678901234567 商品 E
```

これらのファイルは必要に応じて、キーによって結合して使用する。例えば、商品 ID、原価、売価の表が必要な場合、以下の様に tagloopj コマンドを使用する。

```
$ tagloopj key=商品 ID              \
>  SHOHINID_GENKA SHOHINID_BAIKA
商品 ID 原価 売価
1234567890123 100 120
2345678901234 80 100
3456789012345 120 150
4567890123456 30 50
5678901234567 10 30
```

tagloopj コマンドはキー、商品 ID でレコードがソートされている事を前提として動作する。そのため、tagloopj コマンドは $O(n)$ で動作し、使用するメモリも少ない。その他の結合の方法については後述する。

2.3.2 マスタデータの編集

マスタデータのレコードは、新規に作成されたり、更新されたり、削除されたりする。ユニケージではこういった編集処理についても、まずはレベル 1 として受け付け、レベル 2 の形に加工する。この時、キーと編集処理を受け付けた時刻でソートしておく。

例えば、先ほどの商品 ID と売価の対応表ファイル、“SHOHINID_BAIKA” に対する編集処理のレベル 2 ファイルは、以下のような形になる。

```
$ cat SHOHINID_BAIKA.OP
商品 ID 売価 操作 受付時刻
1234567890123 120 削除 201704011023
3456789012345 100 更新 201704011024
3456789012345 130 更新 201704011025
6789012345678 100 新規 201704011026
```

もとのデータに、“操作”、“受付時刻” の二つのフィールドを追加した形式である。

これらの編集処理を実行する基本的な方法は以下のとおりである。まず、awk コマンドを用いて編集対象のデータにもこの二つのフィールドを追加し、tagup3 コマンドを用いて編集処理データとマージする。これは以下の様になる。

```
$ awk 'NR==1{ print $0, "操作", "受付時刻" }
   NR!=1{ print $0, "既存", "000000000000" }' \
   SHOHINID_BAIKA                              |
  tagup3 key=商品 ID - SHOHINID_BAIKA.OP
商品 ID 売価 操作 受付時刻
1234567890123 120 既存 000000000000
1234567890123 120 削除 201704011023
2345678901234 100 既存 000000000000
3456789012345 150 既存 000000000000
3456789012345 100 更新 201704011024
3456789012345 130 更新 201704011025
4567890123456 50 既存 000000000000
5678901234567 30 既存 000000000000
6789012345678 100 新規 201704011026
```

この時、それぞれのキー (商品 ID) の最後のレコードが最新のレコードである。また、最後のレコードの操作が削除である場合、そのレコードは削除される事になる。したがって前の処理に加えて、taggetlast コマンドを使用してそれぞれのキー (商品 ID) の最後のレコードを抽出し、tagcond コマンドを使用して操作が削除であるレコードを除外し、tagdelf コマンドを使用して二つのフィールド、“操作”、“受付時刻” を削除する事で新しいテーブルを作成する事ができる。新しいテーブルは、

“SHOHINID_BAIKA.NEW” などといったファイルに格納する。この処理と、“SHOHINID_BAIKA.NEW” の内容は以下のようになる。

```
$ awk 'NR==1{ print $0, "操作", "受付時刻" }
   NR!=1{ print $0, "既存", "000000000000" }' \
   SHOHINID_BAIKA                              |
  tagup3 key=商品 ID - SHOHINID_BAIKA.OP       |
  taggetlast key=商品 ID                       |
  tagcond '%{操作}!="削除"'                    |
  tagdelf 操作 受付時刻    > SHOHINID_BAIKA.NEW
$ cat SHOHINID_BAIKA.NEW
商品 ID 売価
2345678901234 100
3456789012345 130
4567890123456 50
5678901234567 30
6789012345678 100
```

最後に、mv SHOHINID_BAIKA.NEW SHOHINID_BAIKA コマンドにより、リネームを行ない更新を行なう。直接更新対象のファイルにリダイレクトをせず、一度 “.NEW” といったファイルに書き込む理由は二つある。一つは処理が失敗した際に不正なデータがファイルに書き込まれることを避けるためである。もう一つは新しいデータの書き込み途中に読み込みが発生し、不正なデータが読み込まれる事を避けるためである。mv コマンドによる inode の書き換えは多くの Unix においてアトミックな処理であり、これにより簡易的な排他処理を実現する事ができる。

編集処理を実際に行なうタイミングは基本的にサービスアウト時間である。即座に更新処理の反映されたレベル 3 が必要な場合は、後述のレベル 4 として処理を行なう。

2.3.3 トランザクションデータの整理

トランザクションデータは、その増加傾向に基づき、年毎、週毎、日毎などの単位で、ファイルに格納する。また、マスタと結合して得られる値は削除しておく。例えば、2.1 節で取り上げた例のような形がトランザクションデータであり、“URIAGE.201702”、“URIAGE.201703” のような形で、どの期間のデータが格納されているか分かるようなファイル名を付ける。

2.3.4 トランザクションデータの編集

トランザクションデータに対する編集は、その性質上追記のみである。既存のファイルに追記されるレコードをマージして新しいファイルを作成し、マスタデータと同様に mv コマンドにより更新を行なう。

2.4 レベル 4: アプリケーションデータ

レベル 4 は、インタラクティブシステムなどにおいて素早くレベル 5 を出力するために作成される。そのための加工は、大きく分けて二種類である。

前処理

一つ目はあらかじめ必要とされる形式にデータを加工しておくというものである。例えば、特定の商品について、特定の期間の店舗毎月毎の商品の売上データの推移を参照するシステムでは、2.1 節で取り上げたデータを加工して、以下のようなデータを作成しておく。

```
店舗 ID 商品 ID 商品名 売上月 売上金額 売上個数
001 1234567890123 商品 A 201702 49680 414
001 1234567890123 商品 A 201703 40500 405
001 1234567890123 商品 A 201704 55200 368
```

このように必要な計算を行い、必要なデータを結合しておくことにより、要求に応答する際には絞り込みの処理のみで出力を行なうことが出来るようになる。

分割

もう一つは検索処理など、絞り込みを行なうような処理にたいして、あらかじめファイルを分割して実際に読み書きするデータ量を抑えるというものである。たとえば、先ほどの特定の期間の店舗毎月毎の商品の売上データの推移を参照するシステムでは、一度に必要なデータは特定の店舗の特定の商品についてのデータである。したがって、店舗 ID、商品 ID にもとづきデータを分割してファイルに格納することができる。たとえば、以下のようなファイルツリーに分割される。

- TEN_TSUKI_URIAGE
  - TENPOID.001
    - SHOHINID.XXXXXXXXXX001
    - SHOHINID.XXXXXXXXXX002
    - ...
  - TENPOID.002
  - TENPOID.003
  - ...

まず、データの種類の名前のディレクトリ “TEN_TSUKI_URIAGE” を作成する。さらにその下に “TENPOID.00001” といった店舗コード毎のディレクトリを作成する。それぞれのディレクトリの中に、“SHOHINID.XXXXXXXXXX001” といった名前でファイルを作成し、特定の店舗の、商品 ID の末尾が “001’ である商品の店舗毎月毎の商品の売上データを格納する。こういったファイルツリーを作成するのは、まず経験的に一つのディレクトリ以下に配置されるファイル数は、1000 個以内に抑えられる事が望ましいとわかっているという事が上げられる。また、一つ一つのファイルが均等に分割されることが望ましい。商品 ID のような大きな ID 空間を持つものについては、例のように末尾数桁により分割を行なう事で、比較的に均等に分割を行なう事が可能である。

編集処理の即時反映

レベル 3 において編集処理の基本的な方法を説明した。レベル 3 が更新された場合、レベル 4 もそれに合わせて更

新する必要がある。

前述の基本的な方法はサービスアウト時に実行する場合には十分な実行時間をとる事ができる。しかし、インタラクティブなシステムにおいて編集処理結果をレベル 4 に反映したい場合、応答時間が長くなりすぎることがある。

この場合、“SHOHINID_BAIKA.OP” のような編集処理データをレベル 4 の形式に加工し、この結果をあらかじめ作成されたレベル 4 にかぶせて (`tagup3` して `getlast` して) 更新の反映されたレベル 4 のビューを作成し、これを元にレベル 5 を作成する。

ユニケージの処理は概ね処理データ量に比例する。基本の更新処理に時間がかかるのは、更新対象の全てのデータを読み込む必要があるからである。レベル 5 作成時であれば出力の範囲が明確になるので、編集処理データ、編集対象データを、レベル 5 作成に必要なデータのみに絞り込む事により、処理時間を大きく削減することができる。

2.5 シェルスクリプトによるデータ処理

ユニケージはシェルスクリプトによりデータを処理する。この時注意点として、変数プログラミングをしてはいけないということが上げられる。1.2 節で述べたように、パイプを用いてコマンドにより処理を行なわなければ、性能が得られないばかりか可読性も低下する事がおおい。こんにちの多くのプログラマは命令型言語に馴染みが深いため特に注意が必要である。

以下ではどのようにシェルを用いてデータ処理を行なうか見て行く。

2.6 基本操作

1.1 節でとりあげた松田氏の観察から、企業システムにおけるデータ処理はおおよそ 6 つの処理パターン、1. セレクト、2. ソート、3. サムアップ、4. マッチング、5. 項目間演算、6. 装飾処理、でカバーできる事が分かっていた。Tukubai はこれらの処理パターンを実装したコマンドを核とし、その他業務システムを開発する上で極めて頻繁に出現する処理を、シェルコマンドのセットとして実装したものである。

関係代数の基本演算は、 Tukubai コマンドを使用して書き下す事ができる。以下に関係代数の代表的な演算と、ユニケージにおける記述例を示す。表中の `f1`、`f2` はファイル名であり、`id`、`v1` はフィールド名である。ユニケージにおけるデータファイルは全てキーの値などでソートされている事が前提である。以下の例では第一フィールドがキーであると仮定している。

関係代数	ユニケージ記述例
和 (キー)	`tagup3 key=id f1 f2 | taggetlast key=id`
差 (キー)	`tagjoin0 +ng3 key=id f1 f2 \ 3>&1 1>/dev/null`
交わり (キー)	`tagjoin0 key=id f1 f2`
直積	`tagjoinx f1 f2`
制限	`tagawk '${v1}=="001" f1`
射影	`tagself id v1 f1`
内部結合	`tagjoin1 key=id f1 f2`
外部結合	`tagjoin2 key=id f1 f2`
要約 (合計)	`tagsm2 key=id val=v1 f1`
要約 (数え上げ)	`tagcount key=id f1`

多くのコマンドは入力データがキーについてソートされている事が前提となっている。使用する際には事前にソートを行なう。

このように Tukubai コマンドは関係代数の演算をカバーしている。

2.7 複雑な操作

前節で述べたような基本的な操作を組み合わせる事により、SQL による処理を Tukubai コマンドによる処理に置き換える事が可能である。

例えば、利益率が 3 割以上の商品の、商品 ID、商品名、利益率を出力するような SQL 文は以下のようになる。

```
SELECT SHOHINID_SHOHINMEI.商品 ID, 商品名,
        売価 / 原価 AS 利益率
  FROM ( SHOHINID_GENKA JOIN SHOHINID_BAIKA
           ON SHOHINID_GENKA.商品 ID =
                SHOHINID_BAIKA.商品 ID )
  JOIN SHOHINID_SHOHINMEI
    ON SHOHINID_SHOHINMEI.商品 ID =
         SHOHINID_GENKA.商品 ID
  WHERE 売価 / 原価 > 1.3;
```

これに対応するシェルスクリプトは以下のようになる。

```
tagloopj key=商品 ID SHOHINID_SHOHINMEI \
  SHOHINID_GENKA SHOHINID_BAIKA |
tagawk '
  NR==1{ print $0, "利益率" }
  NR!=1&&%{売価} / %{原価} > 1.3{
         print $0, %{売価} / %{原価} }'|
tagself 商品 ID 商品名 利益率
```

また、より複雑な処理の場合は、一時ファイルに処理を書き出すことによって、段階を追って書き下す。

2.8 BASE トランザクション

ユニケージにより構築されたデータベースシステムは、

2.3.2 節、2.3.4 節で述べたような方法で更新処理を行なう。ここからわかるように、ユニケージにより構築されたデータベースシステムのトランザクション処理は、基本的に BASE トランザクションである。

排他処理を行うための Tukubai コマンドも存在し、ACID トランザクションを実装することも可能ではあるが、著しく性能が悪化するため実装される事は稀である。

3. 評価

ユニケージにより構築されたデータベースの性能を評価するため、ユニケージと MySQL にて同等のデータベースを作成し、同等の処理を行なって処理時間の比較を行なった。性能評価環境は以下のとおりである。

- CPU : Intel(R) Xeon(R) W5580 @ 3.20GHz
  - 8 Cores
- Mem : 47 GiB
- OS : CentOS Linux release 7.2.1511 (Core)
- MySQL : Ver 14.14 Distrib 5.7.18, for Linux (x86_64) using EditLine wrapper
  - 文字コードを UTF-8 に設定した
  - それ以外は `yum` コマンドによりインストールをしたままである。

3.1 検索処理

2.7 節にて取り上げた処理を、10^6 件レコードのテーブルに対して実行し、`time` コマンドにより実行時間を計測した。MySQL へのクエリの送信には、`mysql` コマンドを使用した。テーブルを作成する際には、ユニケージでは “商品 ID” フィールドをキーとしてファイルをソートしておき、MySQL では同フィールドをプライマリキーとしてテーブルを作成した。ユニケージ、 MySQL のいずれについてもこれ以上の高速化の工夫は施していない。`mysql` コマンドはソケットインターフェースを経由する。ユニケージでも実行条件を揃えるため、`ssh` コマンドを利用してネットワーク越しに処理を実行して結果を取得した。この時の 5 回の実行時間の平均値は以下のとおりである。

ユニケージ	MySQL
1.708 秒	10.428 秒

ユニケージにより構築されたデータベースは特段の高速化の工夫を施さずとも、MySQL より高速である事がわかった。

3.2 編集処理

ユニケージにおけるデータベースの編集処理は、2.4 節で述べたように、編集操作が行なわれた時点では実際の編集処理は行なわれず、そのテーブルを利用する参照が行なわれた時点で実際の編集処理が行なわれる。したがって、複数レコードをまとめて編集されることを前提としている。

そこで、10^6 件レコードの “SHOHINID_BAIKA” テーブルに対する 1 件、10 件、 100 件、 1000 件の更新処理を行い、`time` コマンドにより実行時間を計測した。ユニケージの処理時間には、更新処理データを商品 ID と受付時刻によりソートする時間が含まれている。MySQL はデフォルトで自動コミットが有効になっているため、BEGIN クエリ、 COMMIT クエリを使用して複数の UPDATE クエリが同時に処理されるようにする。その他の条件は前節と同様である。

この時の実行時間は以下のとおりである。

	ユニケージ	MySQL
1 件	0.46 秒	0.081 秒
10 件	0.45 秒	0.103 秒
100 件	0.46 秒	0.158 秒
1000 件	0.45 秒	0.21 秒

ユニケージにより構築されたデータベースは、MySQL より遅い事がわかる。

特徴的な点として、ユニケージにより構築されたデータベースは、更新件数が変化しても処理時間が変わっていない。これは更新対象データに対する更新件数が十分すくないため、更新対象データの読み込みに掛かる時間が大半を占めるためである。

ユニケージにより構築されたデータベースにはこういった性質があるため、リアルタイム反映が必要なケースについては、2.4 節で述べたような技法により、読み込むデータ量を減らして処理の高速化を図る。

参考値として、同様の更新処理を施した “SHOHINID_BAIKA” テーブルから、特定の商品 ID のレコードを取り出す場合の処理を、2.4 節で述べた技法に基づいて実装した場合の処理時間を以下に示す。

	ユニケージ (リアルタイム処理)
1 件	0.083 秒
10 件	0.082 秒
100 件	0.081 秒
1000 件	0.082 秒

4. 今後の展開

これまで述べたように、ユニケージに基づくシステムはファイルにデータを格納し、データベースに対する処理についてもファイルに格納することにより処理を行なう。

スケールアウトと Hadoop との比較

このような実装方法であるため、ファイルを転送することにより、データベースに対する操作を他のノードに伝搬することが可能である。ユニケージに基づいて構築されたシステムは、現在の Unix が標準でサポートしている ssh などの機能を利用する事で、容易にスケールアウトする事が可能である。

ユニケージではこの性質を利用して、始めからスケール

アウトを前提とした設計をが行なわれる。また、特に大量のデータを処理するためのアプライアンス製品として、InfiniBand を用いたクラスタ、 usp BOA を開発し発売している。現在、ユニケージシステムの分散処理性能について、現在大規模分散環境での Hadoop との比較を計画している。

POSIX 中心主義

また、シェルスクリプトの移植性を高めるために、POSIX 中心主義というコンセプトを打ち出しの研究を進めている [6]。UNIX の標準規格である、 POSIX に極力のっとったシェルスクリプトを記述することで、移植性、持続性に優れたシェルスクリプトを記述することが可能である。

ものグラミング

さらに、シェルスクリプトの IoT 分野への適用として、ものグラミングというコンセプトを打ち出し研究を進めている [7]。近年では Raspberry Pi に代表される、 Unix 系 OS を搭載した高機能マイコンが利用されることが増えてきている。ものグラミングはこういった高機能マイコンの機能を最大限活用するために、Unix 系 OS が提供する優れた基本機能と、多種多様な POSIX コマンド群を積極的に利用する手法である。

今後も様々な形で Unix 系 OS とシェルスクリプトの可能性を追求して行きたい。

参考文献

[1] Gancarz, M., 桂, 芳尾：UNIX という考え方: その設計思想と哲学，オーム社 (2001).
[2] The IEEE and The Open Group: The Open Group Base Specifications Issue 7 IEEE Std 1003.1, 2013 Edition (2013).
[3] 松田康之：”川下”からの流通情報戦略 「情報武装」革命，オフィス 2020 (1987).
[4] Cheung, S., Nakamoto, M. and Hamuro, Y.: NYSOL: A User-Centric Framework for Knowledge Discovery in Big Data, *International Journal of Knowledge Engineering*, Vol. 1, No. 3 (2015).
[5] 當仲寛哲，山崎裕詞，熊谷 章，熊野憲辰，木ノ下勝郎，山科敦之：ユニケージ原論，ユニバーサル・シェル・プログラミング研究所 (2010).
[6] 松浦智之，大野浩之，當仲寛哲ほか：ソフトウェアの高い互換性と長い持続性を目差す POSIX 中心主義プログラミング，マルチメディア, 分散協調とモバイルシンポジウム 2016 論文集， Vol. 2016, pp. 1327–1334 (2016).
[7] 大野浩之， 森祥寛，北口善明，中村和敬，松浦智之，石山雅三，當仲寛哲ほか：ものづくりのための「ものグラミング」 と実践的教育環境の構築，マルチメディア, 分散協調とモバイルシンポジウム 2016 論文集， Vol. 2016, pp. 1335–1340 (2016).