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　當仲 寛哲

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る。企業システムの開発に 20 年の実績がある。近年のほとんどの企業システムは RDBMS を用いて構築される。全てのデータは RDBMS に格納され、Java などにより開発されたプログラムによりデータを処理する。これに対して、ユニケージにより開発されたシステムは Unix の機能のみにより構築される。これは、ユニケージはデータベースを構築する事ができるからである。全てのデータは Unix ファイルシステムのテキストファイルに格納され、シェルコマンドによってデータを処理する。このシェルコマンドはシェルスクリプトによって起動され、Unix パイプによって連携するものである。本稿では、ユニケージよりどのようにデータベースを構築し、データを操作するか述べ、ユニケージシステムの性能を測定し、類似の製品との比較を行なう。

Unicage is a system development method based on UNIX philosophy and has been applied on business system integration for 20 years. In these days, almost all business system is based on RDBMS. All data is stored in RDBMS and processed by some programs written in programming language such like Java. In other hands, the systems developed by Unicage is based on only Unix functions, because Unicage can build database system. All data is tored in text files of Unix file system and processed by shell commands those are invoked by shell script and cooperating by Unix pipe. In this paper, the authors explain how to construct databese and operate the data bye Unicage, and about comparison between Unicage system and RDBMS based system.

1　はじめに

近年の企業活動、すなわち業務に、情報システムによるサポートは欠かせないものとなっている。これに応えるため、さまざまなパッケージ製品やフレームワークが開発されている。

あるひとつの企業の業務をみた場合、その中には他社との差異のほとんどない業務が存在する。たとえば、法律による規定のある財務会計や労務管理などである。こういった業務をサポートするシステムには、パッケージ製品が最適である。パッケージ製品を導入し、カスタマイズ無しで使用する事で最大の費用対効果を上げる事ができる。

一方で、他社と大きな差異のある業務も存在する。個々の企業の競争力の差を生み出す要因の一つは、企業の経営資源をどのように運用するかの違い、すなわち業務の違いである。そのため、各企業は常に業務に独自の工夫を加え、他社との差別化を図ろうとする。さらにそういった業務は、市場環境の変化に応じて、常に変化し続けることが求められる。

したがって、そういった個々の企業の競争力の差を生み出す業務をサポートするシステムは、パッケージ製品ではカバーする事が難しい場合が多く、フレームワーク、ミドルウェアなどを組み合わせて、スクラッチから構築されることが多い。現代の企業システム開発においては、RDBMS[†1] と Java の組み合わせが

Unicage : Database development method by Unix File System and Schellscript

Nakamura Kazutaka　Tounaka Nobuaki, 有限会社ユニバーサル・シェル・プログラミング研究所, Universal Shell Programming Laboratory Ltd..

†1　Relational DataBase Management System

一般的となっている。そのため、システムを理解するためにはさまざまな製品の知識が必要となっている。したがって、従来手法の学習コストは高いものになっている。

1.1 ユニケージ開発手法

これに対し、ミドルウェア、 DBMS 等を全く使用せず、Unix 上でのシェルスクリプトによるテキスト処理のみによって企業システムを開発する手法が、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 (以下ユニケージ) [3] である。ユニケージは他にも様々な工夫にもとづき、生産性、柔軟性が高く、性能の高いシステムを安価に開発できる事を目指してしており、ユニケージは主に伝票などの文字データ、数値データを処理する企業システムの開発に 20 年の実績がある。

ユニケージはその基礎を UNIX 哲学 [2] におき、KISS 原則の徹底、プログラムの依存関係の排除、など独自の規範を取り入れて発展した、包括的なシステム開発のための規範である。これらは、 UNIX 哲学を実際の企業システム開発に適用し続けた結果、自然と取り入れられたものである。

1.2 ユニケージのカバー領域

その結果ユニケージは、以下のようにシステム開発の様々なフェーズをカバーするようになった。

- システムアーキテクチャ
  - データストアポリシー
  - ノード分散ポリシー
- シェルスクリプトの書き方
  - シェルコマンド利用ポリシー (Tukubai コマンドセット)
- プロジェクト運営手法
  - ドキュメントに対する考え方
  - テストに対する考え方
- システムインテグレーション契約のポリシー

ユニケージでは、 Unix 系 OS のインストールされたシステム上で、シェルスクリプトによってシステムを開発する。シェルスクリプトでは一般的な Unix コマンドのほか、USP 研究所によりユニケージのために開発されたシェルコマンドのセット、Tukubai コマンドセットを利用する。そのため Tukubai コマンドセットやシェルスクリプトの書き方が注目されることが多いが、それ自体はユニケージの一面でしかない。ユニケージにおいて特に重要なのはシステムアーキテクチャであり、ここからその他の規範が派生している。

1.3 本稿の内容

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、類似手法との簡単な比較を行なう。ユニケージは包括的な開発手法であり、カバー範囲は多岐に渡る。本稿ではユニケージのシステムアーキテクチャのうち、特にスタンドアロンのデータベースシステムの構築に必要な部分にのみ焦点を当てて説明する。

まずユニケージシステムのアーキテクチャについて概観し、次にユニケージに特有のデータ処理の方法について見て行く。続いて、 SQL データベースとの比較を行なう。まず機能的な側面の違いについて述べ、次にユニケージシステムの性能を測定する。これに加えてユニケージの関連研究について述べ、最後にユニケージの利点を検討する。

Tukubai コマンドセットには、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 ユニケージシステムのアーキテクチャ

本節ではユニケージシステムのアーキテクチャ(ユニケージアーキテクチャ) について見て行く。まず、ユニケージアーキテクチャが基礎をおく、 Unix システムの要件を述べる。つづいて、ユニケージアーキテクチャについて説明する。

2.1 Unix システム要件

まずは、 Unix システムに対する要件について述べる。ユニケージシステムは現代的な Unix 系 OS のシステム環境に構築される。多くの場合、標準的な Unix 環境に加えて、 Tukubai コマンドがインストールされる。以下では、 Unix 標準のコマンドのほか、パッケージシステムなどでインストール可能なコマンドと、Tukubai コマンドセットのコマンドを総称して、単にコマンドと呼ぶ。

2.1.1 コマンドとその開発指針

ユニケージシステムで用いられるコマンドは Unix 哲学に基づく、単機能のコマンドである。システム開発に伴い必要に応じてコマンドを開発する事もあるが、その場合でも Unix 哲学に基づいて開発する。そのため、ほぼ全ての場合で業務に関するアルゴリズムは含むコマンドが開発されることはない。コマンドは純粋なデータ処理、システムの入出力処理のみを行なう。

2.1.2 サービス、ネットワーク、設定他

Unix 標準のサービスとしては、ユニケージシステムのスクリプトを起動するために crond が、プリミティブなユーザインターフェースとして sshd を通じてのシェルがよく使われる。グラフィカルなユーザインターフェースが必要な場合には、httpd を利用して Web インターフェースを作成することがおおい。本稿ではシェルユーザインターフェースとしてシェルを利用する。

ネットワークに関しては、セキュアなネットワークを利用する事が推奨されている。これはシステムのセキュリティ機能をある程度省略して開発コストを下げることが出来るからである。本稿でも深刻なセキュリティ上の脅威がないことを前提として解説する。その他 RAM ディスクや Unix のセキュリティ機能、NFS などの分散ファイルシステムやその他ミドルウェアなどの利用については、個々のプロジェクトの裁量に任されている。

2.1.3 分散システム

ユニケージアーキテクチャは、本来的に冗長構成の分散システムとして考えられている。これはシステムのハードウェア構成を企業の組織構造に合わせるためである。これにより、部署毎の段階的な業務システム開発やシステム改修を容易にし、対障害性を確保している。しかし、本稿では簡単のため 1 ノードに閉じたスタンドアロンシステムに焦点をあてて解説する。

2.2 構成要素

以降の節ではユニケージアーキテクチャについて述べる。ユニケージシステムの構成要素は以下の 3 つである。

2.2.1 データファイルツリー (ファイル)

第一が処理対象データの格納されたファイルツリーである。ユニケージシステムの内部データは UTF-8 エンコードのテキストでファイルに格納され、基本的に行指向、スペース区切りのテーブル形式である。入力データ、出力データはその限りではない。

通常、一つのテーブルは一つのファイルに格納される。特に応答性能を向上させたい場合などは、ディレクトリツリーを工夫する事で、データ格納のみならず、検索にもファイルシステムを活用する。この場合、一つのテーブルがあるディレクトリ以下のファイルツリーに分割して配置される。

以下ではデータの格納されたファイルもしくはファイルツリーを、単にファイルと呼ぶ。

2.2.2 データ処理スクリプト (スクリプト)

第二はデータを処理するシェルスクリプトである。

スクリプトは通常いくつかのファイルを読み込み、一つのファイルを書き出す。このとき、スクリプトは自身が書き出したファイルを読み込む事はない (スクリプト中で作成され、スクリプトの終了とともに消去されるテンポラリファイルを除く)。これはつまり、スクリプトは状態をもたないという事である。

データを処理するスクリプトはコマンドのみを呼び出し、他のデータを処理するスクリプトを呼び出すことはない。これは後述するデータフローの観点から、システムの構成要素の依存関係を簡潔に保つためである。

以下ではデータを処理するシェルスクリプトを、単にスクリプトと呼ぶ。

2.2.3　スケジューラスクリプト (スケジューラ)

第三は、データを処理するスクリプトを予め定められた順序で起動する、シェルスクリプト、スケジューラスクリプトである。

スケジューラスクリプトはデータを処理するスクリプトの呼び出しを行ない、データの格納されたファイルの読み書きを行なわない (ログなどのファイルは除く)。

スケジューラスクリプトは通常、日毎、月毎などの起動タイミング毎、および後述するシステム階層毎に作成され、crond などから呼び出されるように設定される。

以下ではスケジューラスクリプトを、単にスケジューラと呼ぶ。

2.3　処理モデル

ユニケージシステムはファイルを通じたデータフローシステムである。

ユニケージシステムは、後述する 5 段階のデータ分類にしたがってデータを処理する。入力データはまずファイルに格納され、これをスクリプトによって段階的に処理を行ない、最終的なシステムの出力データを得る。

このとき、それぞれのスクリプトは自身の書き出すファイルより、より出力に近いファイルを読み込むことはない。したがって、データの流れが閉路をつくることはない。

また、それぞれのスクリプトは他のデータ処理スクリプトを呼び出すことはない。これはスクリプトの依存関係を排除するということである。一方で、ファイルは、さまざまなスクリプトから読み出されうる。これは、各スクリプトの間ではファイルにより依存関係が成り立っているということである。モジュール化を行ないたいばあいは基本的に、新しいファイルを作成する、そのファイルを出力するスクリプトを開発することによって行なう。スクリプトから呼び出される、モジュールとなるスクリプトの開発は禁止されている。これによりユニケージはシステムの構成要素の依存関係を簡潔に保っている。

それぞれのスクリプトの起動タイミングはスケジューラによって決められる。最終的な出力データを作成するスクリプトについては、必要となった時に起動されるケースもある。

この性質により、ユニケージシステムは入力データを保存しておく事で、任意の時点でのシステムの状態を再現することができる。

2.4　システム階層

ある程度以上の規模のユニケージシステムには、複数の種類の入力データと、複数の種類の出力データがあり、それぞれの出力データについてデータの流れがある。

この複数のデータの流れが混乱せず、企業の各部署で分担してシステムを開発、更新できるようにするため、ユニケージシステムは 2 階層のシステム階層、5 段階のデータ分類に基づき構築される。

2.4.1　データ基盤系

システム階層の第 1 階層はデータ基盤系である。入力データから段階を追って整理データを作成し、整理データをシステム全体に提供する。

通常データ基盤系用に起動タイミング毎のスケジューラが作成される。データ基盤系用のスケジューラは DATAMASTER などと呼ばれる。

L1:入力データ

データ分類の第 1 段階は入力データである。L1(レベル 1) ファイルなどと呼ばれることもある。

システムに対する入力データは、入力の単位毎にファイルに格納される。たとえば一つの売上伝票の入力は、一つの L1 ファイルに格納される。

入力データはユニケージシステムの外部からやって来るものであり、必ずしもテキストデータには限定されない。

L1 ファイルが存在すればこれ以降の全てのデータは作成出来るので、L1 ファイルは特に厳重に保管される。L1 ファイルは特に数が増えやすい傾向にあるため、適当なタイミングでアーカイブするなどしてファイル数を抑える。

L1 ファイルを取得するスクリプトは、L1GET などと呼ばれることがある。

L2:確定データ

データ分類の第 2 段階は確定データである。L2(レベル 2) ファイルなどと呼ばれることもある。

L2 ファイルは、L1 ファイルをある程度の数まとめ、必要であればテキストのテーブル形式に変換したものである。たとえば、一日分の複数の売上げ伝票の L1 ファイルから、一日分の一つの売上げ伝票の L2 ファイルを作成する。

L2 ファイルを作成するにあたっては、原則データの取捨選択などは行なわない。可能な限り全てのデータをそのままテーブル形式に変換する。

画像データなどテーブル形式への変換が困難なものであり、特に変換の必要性のないデータについては、別途ディレクトリなどを用意して格納しておく。本稿ではこのケースについては取り上げない。

Tukubai コマンドには、L1 ファイルを L2 ファイルに変換するためのコマンドが多数用意されている。例えば Execl ファイルからデータをテキスト形式で読み出す rexce コマンドである。本稿ではその詳細には触れない。

L2 ファイルを作成するスクリプトは、L2MAKE などと呼ばれることがある。

L3:整理データ

データ分類の第 3 段階は整理データである。L3(レベル 3) ファイルなどと呼ばれることもある。

L3 ファイルは、L2 ファイルと過去の L3 ファイルから作成される。参照するファイルは過去の L3 ファイルに限定されるので、データの流れが閉路をなすことはない。

L3 ファイルは後述するアプリケーションの実装をしやすいように整理され、提供される。L3 ファイルはその性質により大きく 2 種類に分類できる。

1 種類目がトランザクション (トラン) である。これは時々刻々と蓄積される種類のデータであり、L2 ファイルから作成される。例えば売上げ伝票トランであり、日毎、地域毎などに別々のファイルに格納される。

2 種類目がマスタである。これはキーに紐づく各種の値のテーブルであり、L2 ファイルと、過去の L3 ファイル (前日など) から、新しい L3 ファイル (当日など) を作成する。例えば、商品の名前、原価、販売価格などの格納されたテーブル、商品マスタである。

L3 ファイルの整理、更新の標準的なテクニックについては後述する。

L3 ファイルの数は必要最小限に抑えられることが望ましいが、一方で必要に応じて自由に新しい L3 ファイルを作成してよい。L3 ファイルはデータベースにおけるテーブルと対比されることが多いが、ユニケージシステムはあくまでデータフローに基づきファイルを処理するシステムである。データのフローは必要に応じて整備される。

L3 ファイルを作成するスクリプトは、L3MAKE などと呼ばれることがある。

2.4.2 アプリケーション系

システム階層の第 2 階層はアプリケーション系である。一つのシステムには複数のアプリケーションが存在する。それぞれのアプリケーションは整理データを参照し、出力データを作成する。また、特に必要な場合に、入力データを直接参照することもある。

それぞれのアプリケーションはファイルの依存関係が発生しないように構築される。そのため、新しいアプリケーションが必要となった場合でも影響調査などは必要なく、気軽にアプリケーションを開発していくことが可能である。

通常それぞれのアプリケーションについて、起動タイミング毎のスケジューラが作成される。アプリケーションのスケジューラは APPMASTER などと呼ばれる。

L5:出力データ

データ分類の第 5 段階は出力データである (第 4 段階については後述)。L5(レベル 5) ファイルなどと呼ばれることもある。

L5 ファイルはシステムの出力データであり、出力の単位毎に作成される。たとえばある店舗の今月の売上げのリアルタイムレポートは、要求のある度に作成される。

L5 ファイルは通常 L3 ファイルから作成される。ただしいくつかの場合、L1 ファイルや、次節で述べる L4 ファイルも利用して作成されることもある。L5 ファイルは標準出力に出力されるケースもある。L1

ファイルを使用する例については後述する。

L5 ファイルは多くの場合、ユーザから要求をうけた時に作成される。大抵の場合作成された L5 ファイルは、ユーザに送信されるなどした後は削除される。

L5 ファイルはユニケージシステムの外部に提供する物であり、必ずしもテキスト形式には限定されない。

Tukubai コマンドには、テーブル形式のデータを様々な形式に変換するためのコマンドが多数用意されている。例えばテーブル形式を Execl ファイルに書き出す wexce コマンドである。本稿ではその詳細には触れない。

L5 ファイルを作成するスクリプトは、L5MAKE などと呼ばれることがある。

L4:アプリケーションデータ

データ分類の第 4 段階はアプリケーションデータである。L4(レベル 4) ファイルなどと呼ばれることもある。

L3 ファイルから直接 L5 ファイルを作成する場合、スクリプトが複雑になったり、処理時間がかかりすぎたりする場合がある。そのような場合に、L5 ファイル作成のリクエスト時に渡される情報なしでも出来るところまで L3 ファイルを処理して、L4 ファイルとして保存しておく。例えば、L5 ファイルとしてある店舗の今月の売上げのレポートを作成する場合、前日の全店舗分の売上げレポートを作成しておく。その上で、リクエストのあった店舗のレポートを抽出して応答する。

L4 ファイルはアプリケーションのキャッシュ的な位置づけである。他のアプリケーションの L4 ファイルを参照してはいけない。また、適切なタイミング (日毎など) で全て消去し、新しいデータを作り直す。

L4 ファイルを作成するスクリプトは、L4MAKE などと呼ばれることがある。

3 ユニケージシステムの実装

本節では、本稿で性能測定に用いるユニケージシステム (例題システム) を例にあげ、ユニケージシステムの実装について概説する。例題システムは小売業のための簡単なシステムであり、アプリケーションとしては、ある店舗のある月の売上げのレポートを作成するアプリケーションをとりあげる。特に断りのない場合、全てのファイルは日毎の営業時間外に更新される。

本節では最低限のシステム構築に必要な部分のみとりあげる。実際のユニケージシステムに必要なログ取得や分散環境での連携に必要な部分などには触れない。スケジューラについては、単にスクリプトを順次起動するものなのでその内容には触れない。スクリプトについては抜粋にて説明し、詳細な書き方については触れない。以下の例では分かりやすさのため、テーブル形式の出力は桁揃えをした形で記述している。

3.1 ディレクトリツリー

ユニケージシステムのファイルとスクリプトは以下のようなディレクトリツリーに格納される。

- DATA：データ基盤系を格納
  - L1：L1 ファイルを格納 ($lv1d)
  - L2：L2 ファイルを格納 ($lv2d)
  - L3：L3 ファイルを格納 ($lv3d)
  - SCRIPT：データ基盤系スクリプトを格納
- APP：アプリケーションを格納
  - REPORT：レポートアプリケーションを格納
    - L4：L4 ファイルを格納 ($lv4d)
    - SCRIPT：レポートアプリケーションのスクリプトを格納
- SHCED：スケジューラスクリプトを格納

このディレクトリツリーはユニケージアーキテクチャの 2 階層 5 段階のシステム階層、データ分類に基づいたものである。データ基盤系のファイルは DATA ディレクトリ以下に、レポート作成アプリケーションは、APP/REPORT ディレクトリ以下に格納される。

これらのディレクトリのパスはスクリプト中ではシェル変数に格納され、たとえば、 DATA/L1 は lv1d などのシェル変数で参照される。以下では各ディレクトリを上記箇条書き中の $lv1d のように参照する。

$lv1d 〜 $lv3d の各ディレクトリ以下には、このシステムで取り扱う 3 種類のデータ、店舗マスタのディレクトリ、 TENPO と、売上データのディレクトリ、URE が作られる。また、 $lv3d 以下には、店舗毎月毎売上データのディレクトリ、 URE_TEN_MON も

作られる。

3.2　テーブルの例

本節では、システムで取り扱うデータについて、例を示しながら説明する。ここで述べるのは論理的なデータの内容である。実際にどのような形でファイルに格納されるかは、次節以降で述べる。

3.2.1　売上データ

まず、整理データのもう一つの種類、トランの例として、売上データをみてみる。

プリミティブな売上げデータは POS レジから出力されるデータほぼそのままのデータであり、店舗 ID、売上時刻、商品 ID、金額、個数からなる以下のようなテーブルである。

```
0001 20170401102340 1234567890123 2280 19
0001 20170401102340 2345678901231 2040 17
0001 20170401012340 3456789012312 2760 23
0001 20170401103052 1234567890123 2280 19
0001 20170401013052 3456789012312 2760 23
0001 20170401103052 4567890123123 2400 20
```

トランザクションデータは、通常キーと日付でソートされた状態で保存される。この売上データの場合、店舗 ID、売上時刻を連結した文字列でソートされた状態で保存される。店舗 ID に紐づく情報は、店舗マスタから取得することができるので、テーブルには含めないことが多い。

店舗毎月毎売上げデータ

売上げデータのようなデータは、通常ある程度加工されて扱われる事が多い。例えば、売上げデータであれば、店舗毎月毎売上げデータのように加工されて出力される。

月毎店舗毎売上げデータは、店舗 ID、売上月、金額からなる以下のようなテーブルである。

```
0001 201611 3680187304
0001 201612 4080081360
0001 201701 4971217222
0001 201702 3386136033
0001 201703 4760390169
```

店舗毎月毎売上データの場合も、店舗 ID、売上月を連結した文字列でソートされた状態で保存される。

3.2.2　店舗マスタ

次に、整理データの一種、マスタの例として、店舗マスタをみてみる。今回、店舗マスタは、店舗 ID、店舗名、電話番号から成る以下のようなテーブルである。

```
0001 千代田区店 0311111111
0002   中央区店 0322222222
0003     港区店 0333333333
0005   新宿区店 0355555555
0006   文京区店 0366666666
```

マスタはキーでソートされた状態で保存される。この店舗マスタの場合、店舗 ID でソートされた状態で保存される。

店舗マスタ操作データ

多くの場合、マスタデータに対する入力データは、マスタそれ自体ではなく、マスタに対する操作のデータである。例えば、店舗マスタの操作のデータは、店舗マスタに操作と操作時刻を追加した以下のようなデータである。

```
0001 千代田区店 0311111111 削除 20170401102555
0002   中央区店 0322224444 更新 20170401102326
0002   中央区店 0322223333 更新 20170401102401
0003     港区店 0333334444 更新 20170401102433
0004   台東区店 0344444444 作成 20170401102624
```

マスタ操作データには、レコード作成、もしくは更新の場合は、キーで指定されるレコードの新しい内容が格納される。レコードの削除の場合は、意味を持つのはキーのみである。

マスタ操作データはキーと操作時刻でソートされた状態で保存される。この店舗マスタ操作データの場合、店舗 ID と操作時刻を連結した文字列でソートされた状態で保存される。

店舗毎月毎売上げレポート (出力データ)

出力データは他システムや人の目に触れる形で提供される。例題システムでは、店舗毎月毎売上げレポートが出力であり、これは、店舗 ID、店舗名、売上月、売上金額からなる以下のようなデータである。

```
0001 千代田区店 2016/11 3,068,087,304
```

最終的に人の目に触れるものであるため、店舗 ID に対して店舗名が付加され、売上月は YYYY/MM

の形式で、売上金額は商業文書の習慣に則り 3 桁毎にカンマを打っている。

以降では、5 段階のデータを作るための処理について述べる。

3.3 L1GET:入力データの処理

L1 ファイルは、ユーザがターミナルや Web をインターフェースを通じて作成するか、外部システムから受け渡される。これを受け取り、ディレクトリツリーの中に正しく配置するのが、L1GET の処理である。

入力データは入力の単位毎にファイルに格納される。

入力データは様々な形式を許容する。ただし例題システムでは簡単のため、3.2 節で述べたテーブルと同じ形式のデータを入力とする。例えば、売上げデータの L1 ファイルは以下のようになる。

```
$ cat $lv1d/URE/0001/20170401102340.923 |
0001 20170401102340 1234567890123 2280 19
0001 20170401102340 2345678901231 2040 17
0001 20170401012340 3456789012312 2760 23
```

ここでは店舗 ID のディレクトリを作成し、ファイル名を <YYYYMMDDHHMMSS>.<プロセス ID> のようにして、異なる入力でファイル名が重複しないようにしている。

この例では、3 つの商品が同じタイミングで購入されたので 3 レコード同時に入力されている。

L1 ファイルは入力のタイミングで作成されるため、日毎に作成されるとは限らない。

L1GET の処理の詳細は入力インターフェース毎に異なるので、本稿ではその詳細には触れない。

3.4 L2MAKE:確定データの処理

L2 ファイルは L1 ファイルをテキストに変換してある程度まとめたものである。例題システムの L1 ファイルは、前節で述べたように 3.2 節で述べた形式であるので、テキストへの変換は必要ない。

例えば、ある日の売上げデータの L1 ファイルから L2 ファイルを作成する処理は、以下のようになる。

```
echo $lv1d/URE/????/20170401??????.* |
xargs cat > $lv2d/URE/20170401
```

1 行目で 4 月 1 日の全ての売上 L1 ファイルの名前を、シェルのワイルドカード展開で取得している。2 行目でファイルを連結して出力している。

シェルのワイルドカード展開を行なった時点で、ファイル名は店舗 ID、売上時刻でソートされるため、cat コマンドにより連結された出力は店舗 ID、売上げ時刻でソートされた状態になっている。

マッチするファイル数が多い場合、コマンドの引数の上限をオーバーしてしまう事がある。xargs を使用するのは、これを回避するためである。

L2MAKE はファイルのオープンクローズをともなうため、コストのかかる処理である。L1 ファイルの分量が多いばあい、営業時間内であっても適宜 L2MAKE を実行する事が望ましい。

3.5 L3MAKE:整理データの処理

整理データはトランとマスタとで異なる形でファイルツリーに格納される。また、更新の方法も異なる。

以下ではそれぞれのファイルツリーの形、更新の方法について述べる。

3.5.1 トランザクションの整理

トランは時々刻々と蓄積される種類のデータである。トランのデータ量は時間の経過とともに増大する。1 ファイルあたりのデータ量が多くなり過ぎると性能面で悪影響があるので、これを抑えるため、日毎、地域毎などの単位で別々のファイルに格納される。例題システムの売上データは、例えば月毎のファイルに分割して格納される。2017 年 4 月のデータであれば、L3 ファイルのパスは以下のようになる。

```
$lv3d/URE/201704
```

このデータ分割の指針は、あつかうデータ量に応じて変更する。

L3 ファイルの内容は 3.2.1 節で述べたように、ソートされて格納される。

3.5.2 トランザクションの更新

トランに対する更新は、その性質上追記のみである。当日分の L2 ファイルを前日の L3 ファイルに追記する形で、当日の L3 ファイルが作成される。

例えば、売上データの当日分の L2 ファイルから、当日分の L3 ファイルを作成する処理は以下のようになる。

```
up3 key=1/2 $lv3d/URE/201704 \
    $lv2d/URE/20170401 > $lv3d/URE/201704.new
mv $lv3d/URE/201704{.new,}
```

1 行目〜2 行目は up3 コマンドを使用して key=1/2 オプションにより、第 1 フィールドの店舗 ID と第 2 フィールドの売上時刻をキーとして前日分の L3 ファイルと当日分の L2 ファイルをマージしている。3 行目では 2 行目で書き出されたマージした内容を、mv コマンドでもとのデータに上書きをしている。

加工データの更新

店舗毎月毎売上げデータのように、加工されたデータの場合であっても、基本的な方針は同じである。

例えば、売上データの当日分の L2 ファイルから、当日分の店舗毎月毎売上げデータの L3 ファイル、$lv3d/URE_TEN_MON/2017 を作成する処理は以下のようになる。

```
self 1 2.1.6 4 $lv2d/URE/20170401 |
up3 key=1/2 $lv3d/URE_TEN_MON/2017 -|
sm2 1 2 3 3 > $lv3d/URE_TEN_MON/2017.new
mv $lv3d/URE_TEN_MON/2017{.new,}
```

1 行目は self コマンドの引数により第 1 フィールド、第 2 フィールド、第 4 フィールドを選択しており、特に第 2 フィールドについては 2.1.6 のように指定することで、YYYYMMDDHHMMSS 形式の時刻から YYYYMM の月部分のみを取り出している。2 行目は先ほどと同様に up3 コマンドによりマージを行なっている。3 行目は sm2 コマンドにより、第 1 フィールドから第 2 フィールドをキーとして、キーが同じレコードの第 3 フィールドの値を合計している。

これにより、3.2.1 節で述べたようなテーブルを作成できる。

3.5.3 マスタの整理

多くの場合、マスタは少数のキーに対して沢山の値が紐づけられたデータである。時には数百種類の値が紐づけられる事もあるが、個々の処理にはその全てを用いるわけではない。また、多くのフィールドを含むレコードはデータ量が多く、可読性も下がる。

こういった問題を解決するため、マス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される。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。

例題システムの店舗マスタは、店舗 ID をキーとするテーブルである。以下のようなキーと値を組みとする 2 つのファイルに分割される。

- $lv3d/TENPO/ID_NAME : 店舗 ID, 店舗名
- $lv3d/TENPO/ID_TEL : 店舗 ID, 電話番号

例えば、 $lv3d/TENPO/ID_NAME の内容は以下のとおりである。

```
0001 千代田区店
0002   中央区店
0003     港区店
0005   新宿区店
0006   文京区店
```

こういった分割されたファイルを利用する際には、以下の様に loopj コマンドを用いて必要な値を連結して使用する。

```
$ loopj num=1 $lv3d/TENPO/ID_{NAME,TEL}
0001 千代田区店 0311111111
0002   中央区店 0322222222
0003     港区店 0333333333
0005   新宿区店 0355555555
0006   文京区店 0366666666
```

loopj コマンドは同じキーのレコードを突合わせて、一つのテーブルを作成する。num=1 はファイルの先頭 1 フィールドをキーと解釈するという意味である。この例では値は 2 つであったが、より多くの値がある場合でも同様に記述して、必要な値のみからなるテーブルを作成する。

3.5.4 マスタの更新

マスタに対する更新は、マスタ操作データを通じて行なう。前日分の L3 ファイルに、当日分のマスタ操作データの L2 ファイルを重ね合わせて、当日分の L3 ファイルを作成する。

ある日のマスタは、その日までのマスタ操作データ全てから作成することが可能である。指定された日までのマスタ操作データについて、各キーについて最新のレコードを抽出し、削除操作のレコードを除いた物がその日のマスタである。

しかしマスタ操作データの蓄積量が多くなると、こ

れでは処理時間がかかり過ぎる。これを避けるために、直前までの更新データの蓄積である、直前の L3 ファイルを利用して新しい L3 ファイルを作成する。

例えば、店舗マスタ操作データの当日分の L2 ファイルが、$lv2d/TENPO/TENPO_OP.20170401 であるとすると、当日分の L3 ファイルを作成する処理は以下のようになる。

```
$ loopj num=1 $lv3d/TENPO/ID_{NAME,TEL} |
> up3 key=1 - $lv2d/TENPO/TENPO_OP.20170401 |
> getlast key=1 |
> delr 4 削除 |
> self 1/3 > $lv3d/TENPO/new
self 1 2 $lv3d/TENPO/new > $lv3d/TENPO/ID_NAME
self 1 3 $lv3d/TENPO/new > $lv3d/TENPO/ID_TEL
rm $lv3d/TENPO/new
```

1 行目〜5 行目までが当日分のテーブルを作成する処理である。6 行目、7 行目ではフィールドを選択してキー、値の形式のファイルを作成して上書きしている。8 行目では中間ファイルである当日分のテーブルを削除している。

当日分のテーブルを作成する手順は以下のとおりである。1 行目で loopj コマンドを使用して前日分のテーブルを作成している。2 行目で up3 コマンドを使用して key=1 オプションにより、第 1 フィールドをキーとして前日分のテーブルと当日分のマスタ操作データをマージしている。このとき、ファイル間で同じキーのレコードは引数で指定した順序でマージされる。3 行目で getlast コマンドを使用して key=1 オプションにより、第 1 フィールドをキーとして、最後に現れるレコードを出力している。これによりそれぞれのキーの最新の操作レコード (操作の無い場合はもとのレコード) が出力される。4 行目で delr コマンドを使用して第 4 フィールド、操作が削除であるレコードを削除している。5 行目で self(SELect Field) コマンドを使用して、引数の 1/3 により、元のテーブルに必要な第 1 フィールドから第 3 フィールドまでを選択して、中間ファイルである$lv3d/TENPO/new に書き出している。

3.6 L4MAKE:アプリケーションデータの処理

L4 ファイルはアプリケーション毎に作成される。例題システムのアプリケーションは、3.2.2 節で述べた店舗の月毎の売上げのレポートを作成するアプリケーションである。

L4 ファイルは特に応答時間が短いことが要求されるインタラクティブシステムなどにおいて、L5 ファイル作成のための応答時間を短縮するためのものである。そのための加工の処理は大きく分けて、前処理と分割の二種類である。

以下ではこの 2 つの方法について述べる。

3.6.1 前処理

L5 ファイルは L3 ファイルを加工して作られる。前処理は、L3 ファイルを可能な限り L5 ファイルの形式に近づけておくことである。

例題システムのアプリケーションの場合、3.2.2 節で述べた出力データの形式に近づけることであり、この処理は以下のようになる。

```
join2 key=1 $lv3d/TENPO/ID_NAME \
$lv3d/URE_TEN_MON/2017 > $lv4d/2017
```

1 行目〜2 行目は join2 コマンドにより、店舗毎月毎売上トラン$lv3d/URE_TEN_MON/2017 に、店舗名の格納された店舗マスタ、$lv3d/TENPO/ID_NAME を突き合わせている。key=1 はトランの第 1 フィールドをキーと見なすという意味である。マスタのほうは、常に先頭から指定された数のフィールドをキーと見なす。

$lv4d/2017 の内容は以下のようになる。

```
0001 千代田区店 201611 3680187304
0001 千代田区店 201612 4080081360
0001 千代田区店 201701 4971217222
0001 千代田区店 201702 3386136033
0001 千代田区店 201703 4760390169
```

月の形式の変換と、金額へのカンマの挿入は、L5 ファイル作成時に行なう。

このように事前に加工を済ませておく事で、L5 ファイルの要求を受けた時の処理量を減らしておく。

3.6.2 分割

前処理でみたように、L4 ファイルによる応答高速化の基本戦略は、全ての回答をあらかじめ作成して

おくというものである。そのため、全体としての L4 ファイルのサイズが大きなものになることがある。

分割は、要求時に与えられるパラメータで L4 ファイルをあらかじめ分割しておく事である。この処理は、前節の処理結果を利用して記述すると以下の様になる。

```
keycut $lv4d/%1.2017 $lv4d/2017
```

1 行目で、keycut コマンドを利用して、ファイル $lv4d/2017 を店舗毎、月毎のファイルに分割している。例えば、 $lv4d/0001.2017 には、店舗 ID 0001 の、2017 年のレコードのみが格納される。実際のスクリプトではほとんどの場合前節の処理と合わせて一つのパイプラインで記述される。

書き出しファイル名は、ディレクトリツリーであることもある。例えば、$lv4d/0001/2017 のような形である。これはレコードの探索木をディレクトリツリーを利用して実装するテクニックである。このようにする事で、L5 ファイルを要求された場合、店舗 ID で絞り込まれたデータを読み出すこととなり、応答時間の短縮が見込まれる。

ただし、あまり細かいファイルを大量に作成しても、性能の劣化が発生することがある。分割の目安としては、1 ファイルが 500MBi を越えたあたりから、ディスクに格納されたファイルであれば読み出し時間が 1 秒を越えるので、こういった分割を考える。

分割の方法には様々なパターンがあり得る。たとえば、店舗 ID が連番振られている場合は、店舗 ID の下 N 桁によりファイルを分割する、等である。これにより各ファイルのサイズが均等にならされると期待できる。

3.7 L5MAKE:出力データの処理

前節までで作成された L4 ファイルから、最終的な出力である L5 ファイルを作成する。例えば、店舗 ID0001 の月 201704 の L5 ファイルを作成する処理は以下のようになる。

```
selr 3 201704 $lv4d/0001.2017 |
dayslash yyyy/mm 3 |
comma 4
```

1 行目では selr コマンドを利用して、店舗 ID0001 のファイルから第 3 フィールドが 2017/04 であるレコードを抽出している。実際には店舗 ID や月はシェル変数に格納されるなどして与えられるので、適当な方法で加工してこの処理を実行する。2 行目は dayslash コマンドにより日付の加工を行なっている。引数で第 3 フィールドを指定し、 YYYYMM 形式の日付を YYYY/MM 形式に加工している。3 行目は comma コマンドにより、商業文書の慣習に従った金額の加工を行なっている。引数で売上金額のはいった第 4 フィールドを指定して、3 桁毎にカンマを挿入している。

これにより、最終的な出力が得られる

3.7.1 リアルタイム処理

例題システムのファイルは、日毎の営業時間外に更新される。一方で営業時間内に、その日の売上を加算したデータを知りたいという場合もある。しかし、新しい L1 ファイルが到着する度に、L4 ファイルまでを構築しなおすのでは、処理時間がかかり過ぎる。

リアルタイム処理はそのような要求があるばあいに、もとの L4 ファイルに含まれていない L1 ファイルの内容を反映して、L5 ファイルを作成する処理である。例えば、2017 年 4 月 2 日に、店舗 ID0001 の月 201704 の、当日分のデータを反映したレポートを作成する処理は以下のようになる。

そのような場合の処理は以下のようになる。

```
{
selr 3 201704 $lv4d/0001.2017

cat $lv1d/URE/0001.20170402??????.* |
self 1 2.1.6 4 |
join2 key=1 $lv3d/TENPO/ID_NAME
} |
sm2 1 3 4 4 |
dayslash yyyy/mm 3 |
comma 4
```

1 行目〜7 行目までグルーピングを用いているが、4 行目〜6 行目、8 行目を除いた場合、前述の L5 作成処理と同じである。

4 行目〜6 行目は、店舗 ID 0001 の 2017 年

4 月 2 日分のデータを読み出して、L4 データ `$lv4d/0001.2017` と同じ形式に加工している。1 行目〜7 行目までグルーピングを用いているので、前日に作成された 4 月の売上げデータと当日分の売上げデータが、8 行目のコマンドの入力となる。

8 行目では、`sm2` コマンドを用いて、前日までのデータに当日のデータを加算している。最後に、9 行目、10 行目でデータの見た目を加工して出力している。

4 ユニケージシステムの特に留意すべき特徴

ユニケージシステムはファイルを通じたデータフローシステムである。そのため、データベースに対する読み書きを基礎におく RDBMS とは、大きく異なる部分も多い。本節ではそのような特に留意すべき特徴について述べる。

4.1 排他制御

RDBMS における排他制御はレコード等、データベース上のオブジェクトに対する読み書きを制御するものである。

ユニケージシステムにおける排他制御は、二つのスクリプトが同じ一つのファイルに対して同時に読み込み、書き込みを行なわないようにするためのものである。これは、書き込まれている最中のファイルを読み込んだ場合、不完全なデータを得てしまうためである。

ユニケージシステムはこれに対して、スケジューリングと OS アトミック処理により排他制御をあつかう。

Tukubai コマンドには排他区間を作成するためのコマンドも存在するが、捕捉のしにくいバグや、性能の低下を引き起こすため、可能な限り使用を回避する。このコマンドについては本稿では触れない。

4.1.1 スケジューリングによる排他制御の回避

まず、ユニケージシステムではスケジューリングにより排他制御を回避する。スケジューラによってスクリプトが正しい起動順序で起動されるならば、排他制御の必要はなくなる。

これは例えば、日次で更新されるファイルがある場合に、そのファイルを作成するためのスクリプトが必ず、L1GET → L2MAKE → L3MAKE → L4MAKE の順で起動されるという事である (L5MAKE はユーザから要求を受けたときに実行される)。それぞれのスクリプトが段階を踏んで起動されるため、同じ一つのファイルが同時に読み書きされることはなくなる。

4.1.2 OS アトミック処理による排他制御の実現

スケジューリングにより排他制御を回避出来ない場合もある。たとえば、3.7.1 節で述べたリアルタイム処理を行なうようなシステムの場合である。そのようなシステムの稼動中に作成される L1 ファイルは、L5 ファイルを要求される任意のタイミングで参照される。ここで、L1 ファイルの格納されるディレクトリに直接データをリダイレクトした場合、作成途中のファイルが参照されてしまう可能性がありうる。

このときは、 OS のアトミックな処理を利用して排他制御を実現する。Unix ファイルシステムにおいて、同じパーティション上での rename(2) システムコールはアトミックに処理される。これはつまり、あるディレクトリに内での、“`mv file.new file` ”というコマンドは、アトミックに処理されるということである。この性質を利用し、ファイル単位での排他制御を行なう。

4.2 BASE トランザクション

ここまでからわかるように、ユニケージシステムは、基本的に BASE トランザクションである。分散システムとして構築されたユニケージシステムがそうであるのはもちろん、スタンドアロンシステムであっても、LV1 ファイルが到達するタイミングによって、一時的な不整合が発生する可能性がある。

排他区間を作るコマンドを使用し ACID トランザクションを実装することも可能ではある。しかし前述のような問題があり、実装される事は稀である。

5 ユニケージシステムの性能測定

筆者らは 3 節で解説した例題システムの性能を測定した。

5.1 測定方法

性能測定に際しては、例題システムのレポートを生成する処理を行い、`time` コマンドを用いて処理時間を測定した。ユニケージシステムについては、L1 ファイルから L4 ファイルまでを生成する日毎の処理 (日次更新処理) と、要求に応じて L5 ファイルを生成する処理 (リアルタイム処理) の時間を測定した。

実際に実行されるスクリプトは以下のとおりである。

- 日次更新処理
  - `L2MAKE.URE` : 売上データの L2MAKE
  - `L3MAKE.URE_TEN_MON` : 店舗別月別売上データの L3MAKE
  - `L4MAKE.URE_TEN_MON` : 店舗別月別売上レポートの L4MAKE
- リアルタイム処理
  - `L5MAKE.URE_TEN_MON` : 店舗別月別売上レポートの L5MAKE(リアルタイム処理)

`L5MAKE.URE_TEN_MON` には、3. 7. 1 節で述べたリアルタイム処理である。それ以外の各スクリプトの内容は、3 節で解説したものとと同等である。

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。

ユニケージシステムのデータは全てファイルに保存される。一方で Linux ファイルシステムは読み込んだファイルをメモリ上にキャッシュする。このため同じファイルを読み込む処理を短い時間内に複数回実行すると、2 回目以降が高速になる傾向がある。今回は全て 11 回の測定を行ない、始めの 1 回を除外した測定結果の平均を取った。

システムを適用する小売りチェーン店の規模を、小規模、中規模、大規模と想定し、それに合わせて以下の 3 通りのデータ量で測定をおこなった。大規模チェーン店のデータ量はイオングループのおおよその規模に倣った。一人当たり購入点数は 10 とした。

規模	店舗数	来客数 人/日	売上データ レコード/日
小規模	10	150	$15 * 10^3$
中規模	100	750	$75 * 10^4$
大規模	1000	4500	$45 * 10^6$

例題システムのデータ量は、小規模から中規模では 50 倍、中規模から大規模では 60 倍となっている。

ユニケージシステムはとりあつかうデータ量に合わせてデータファイルの分割方法を変える。また、このケースでは大規模以上の場合特に、L1 ディレクトリのファイル数が増大するため、営業時間内に L2MAKE を実行することが望ましい。しかし、今回は比較のため、分割方法や L2MAKE 実行タイミングの調整はおこなわなかった。

5.2 結果と考察

測定の結果は以下のとおりである。

規模	日次更新処理	リアルタイム処理
小規模	0.14 秒	0 秒
中規模	1.71 秒	0.01 秒
大規模	103.91 秒	0.09 秒

例題システムの日時更新処理は、小規模から中規模では約 12.2 倍に、中規模から大規模では約 60.7 倍になっている。また、処理時間の大部分を L2MAKE 処理が占めていた。

中規模から大規模での処理時間の増加がデータ量の増加に一致している。これは、店舗毎月毎売上げデータの L3 ファイルを更新するさいに、その日の売上データの L1 ファイルを全て読み込むためと思われる。

小規模から中規模での処理時間の増加はそれほどではないのは、小規模の場合はデータ処理時間そのものよりも、プロセスの起動と終了のための時間が大半をしめているためであると思われる。

応答時間についてはほぼ変化がなかった。小規模のケースで 0 秒となっているのは、`time` コマンドでは測定できなかったということである。これは大規模のケースであっても、L4 ファイルの 1 ファイルあたり

のデータ量はたかだか 12 レコードであり、また、当日分の L1 ファイルも最大で 4.5 万レコード程度と少数であったためと思われる。

6 ユニケージの関連研究と定性的比較

本節ではユニケージの関連研究を見て行く。そのうちいくつかとは、ユニケージとの定性的な比較を行なう。

6.1 POSIX コマンドとユニケージ

ユニケージはテキストファイルをデータベースとして用いる手法である。実は POSIX コマンドにもそのような使い方の萌芽をみることができる。

まず、 Unix version 7 から登場した `join(1)` コマンドである。これは SQL における join と同等の処理を行なうコマンドである。また他にも、`cut(1)`、`paste(1)` などテーブル形式のカラムを操作するコマンドや、`sort(1)` コマンドの`-k` オプションや`-m` オプションなど、テーブル形式のファイルの基本的な操作に必要なコマンドが一通り揃っている。

こういったコマンド群に加え、 awk(1) コマンドが存在するため、Tukubai コマンド無しでもユニケージシステムを構築することが可能である。

6.2 ファルマにおけるシステム開発と NYSOL プロジェクト

ユニケージが特に影響をうけたプロジェクトとして、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、ファルマにおけるシステム開発がある。同社の創業者の一人、松田康之氏は独特のシステム開発手法を編み出していた [4][†2]。

これは、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 (RPG 言語) を用いて開発し、必要に応じて書き捨てるという考え方であった。

現在ファルマでの成果の一部は NYSOL プロジェクト [1] に受け継がれ、公開されている。NYSOL プロジェクトでは KDD(Knowledge Discovery in Database) プロセスに基づき処理を組み立てる。しかしながら、業務システムを開発するためのシステムアーキテクチャは提供されていない。

6.3 POSIX 中心主義

これまでの手法は、様々な言語やミドルウェアなどの製品を用いる手法が主流であった。こういった製品は変遷が激しいことが多く、これらの製品を利用して作成したシステムの寿命は短い。

POSIX 中心主義は、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を目的とする [5]。POSIX の変遷、各 Unix 系 OS の POSIX 準拠状況などを調査している。

ユニケージもほぼ POSIX 準拠のスクリプトを記述する。そのため、移植性、持続性にすぐれたシステムを開発することが出来る。

6.3.1 クラスタ処理

ユニケージはファイルを通じたデータフローシステムである。そのため特段のミドルウェア等なしでも、ファイルコピーによって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。ユニケージはスケールアウトがしやすいシステムである。USP 研究所では、ビッグ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。

また、ユニケージシステムの分散処理性能について、現在分散環境での Hadoop との比較を行なっている。

7 ユニケージの利点

最後に、本稿でみたユニケージの説明と性能測定から説明可能な、ユニケージの利点について述べる。

7.1 高い性能が得やすい

5 節でみたように、ユニケージシステムは数千億レコードの処理に対しても 2 分未満で処理をすることが出来た。また、特に応答性能が求められるシステムについても、高い性能を示した。大規模ケースのデータ量は日本国内有数の大規模小売りチェーン店から

†2 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

取ったものだが、そのような企業のデータ処理に対しても十分対応可能な性能が得られると言える。

また、この性能はユニケージの標準的な技法に基づいて構築されたシステムによって得られたものである。ユニケージシステムは容易に高い性能を得る事ができる。

7.2 学習コストが低い

ユニケージシステムは学習コストが低い。既存手法は、NYSOL プロジェクトのように知識発見のためのパッケージであったり、あるいは RDBMS や NoSQL 製品と Java のように、複数の製品を組み合わせてシステムを開発する必要があった。これに対してユニケージはユニケージアーキテクチャとシェルスクリプトの知識のみで、業務システム全体を理解し構築することが可能である。

7.3 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、ユニケージシステムはデプロイや移植が容易である。

RDBMS や NoSQL を利用したシステムの多くは、複雑な設定を必要とするためデプロイは必ずしも容易とは言えず、また移植性もまちまちであるが容易でない事がおおい。

一方で、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、UNIX 哲学を基礎におくユニケージもまたその方針を受け継いでいる。既に動作しておりデータが蓄積されたユニケージシステムを新しい環境に移植する際には、単純にファイルとコマンドをコピーし、コマンドへのパスを通すだけでよい。

7.4 バグの少ないシステムを容易に開発できる

ユニケージシステムはバグが発生しにくい。シェルのパイププログラムは状態をもたず、また、ユニケージアーキテクチャのスクリプトも状態を持たない。これは関数プログラミングの関数と同様の性質である。この性質により関数プログラミングはバグの少ないシステムを容易に開発できるという利点がある。ユニケージも同様にバグの少ないシステムを容易に開発できる。

8 おわりに

ユニケージは他にも様々な工夫にもとづき、生産性、柔軟性が高く、性能の高いシステムを安価に開発できる事を指向している。今後もユニケージと他の手法との定性的、定量的な比較を様々な側面から行いたい。また、コマンドの開発によるユニケージの適用可能分野の拡大や、さらなる改良に取り組みたい。

参 考 文 献

[1] 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).

[2] Gancarz, M., 桂, and 芳尾: *UNIX* という考え方*:* その設計思想と哲学, オーム社, 2001.

[3] ユニバーサル・シェル・プログラミング研究所: 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.

[4] 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.

[5] 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(2016), pp. 1327–1334.