データ解析型プロジェクトにおけるバージョン管理手法の検討

九州工業大学　○李林　服部祐一　井上創造

Study of Version Management Method for Data-analytic Project

Lin LI, Yuichi HATTORI and Sozo INOUE, Kyushu Institute of Technology

Abstract: In this research, we introduce a version management method for data-analytic project. Our aim is to automatically determine whether the data needs to be update, to improve the efficiency of version management and quality of project. In this paper, we describe an algorithm to realize it.

1. はじめに

近年，Web 技術とセンサ技術の発展により，大量のデータが収集され，解析されるようになった．また解析には複数の段階があり，その度に計算結果が生成される．これらのデータをバージョン管理することができれば，何度も同じ計算をする必要がなくなる．しかし，データ解析型プロジェクトでは，データとデータ，解析プログラムとデータなどがお互いに独立の関係ではない．それらの間には様々な依存関係がある．

よって，自動的にデータやプログラム更新を判断できるシステムが必要となる．本研究では上記の問題に対するバージョン管理の方法を検討し，データフローの視点から，自動的に更新判断するアルゴリズムを提案する．

本論文は 4 つの章からなる．以下では，2 章で関連研究を述べ．3 章で定式化と管理方法を述べ．4 章でまとめる．

2. 関連研究

2.1. 大規模行動情報収集システム「ALKAN」

我々は人間の行動を客観的に計測するための行動解析システムの構築を目標としている．行動解析には教師データとなるデータが必要となり，そのために多くの行動情報が必要になる．我々は，大量な行動情報を収集するシステムとしての大規模行動情報収集システム「ALKAN」を開発した．ALKAN では，iPhone や iPod Touch などの携帯情報端末と行動情報を収集するためのサーバを用い大量の行動情報を収集することを可能としている[1]．また，実際に約 4 万件の行動情報を収集しており，日々膨大するデータに対して，より良いバージョン管理方式を検討しなければいけない．

2.2. バージョン管理システム

現在の主流のバージョン管理システムは主に下記の 2 種類がある．

1) 分散型バージョン管理システム

　分散型バージョン管理システムとは，各開発者がローカルにリポジトリを持つ分散型のアーキテクチャを採用し，必要に応じてリポジトリ間で変更点をやり取りすることによってソースコードなどの管理行う．代表的ものは Git[5]，Mercurial[6]などがある．

2) 集中型バージョン管理システム

　集中型バージョン管理システムとは 1 つのリポジトリにすべての開発者がアクセスする集中型アーキテクチャを採用している．代表的物は Concurrent Versions System（CVS）[7]，Subversion（SVN）[4]などがある．

我々の ALKAN プロジェクトでは，集中型バージョン管理システムである SVN を使って管理している．SVN では，時間と伴に変化するファイルやディレクトリを管理することができる．また，古いバージョンのデータに戻したり，変更履歴を確認した

りすることもできる．SVN の利用形態を図 1 に示す．SVN では，開発者は自分の作業スペースでファイルに対して追加, 修正, 削除などの操作を行う．そしてリポジトリにコミットする．コミットとは作業コピーの変更点をリポジトリに送ることである。アップデートとはリポジトリにある変更を作業スペースに反映することである．

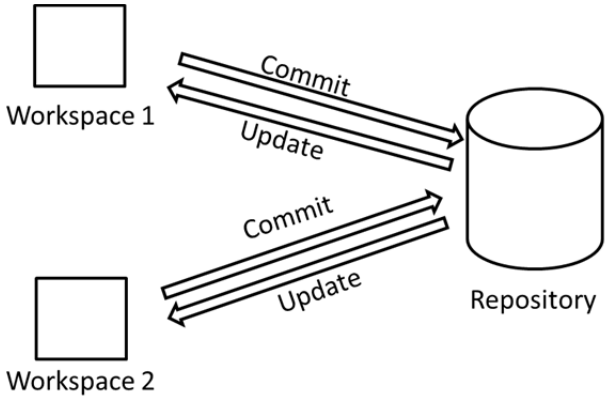

図 1 Subversion のワークフロー

しかし，一般的なバージョン管理ではプログラムのソースコードを管理するが，データ解析プロジェクトでは，プログラムのソースコードに加え，解析に用いるデータも管理する必要がある．SVN を用いてもこれらを管理することはできるが，バージョンによるデータとプログラムの依存関係を把握することが困難である．この問題を改善すれば，開発の効率とプロジェクトの品質を大幅に向上することができる．

3. 管理手法の提案

本章では，自動的にデータの更新条件を判定できる管理手法のアルゴリズムについて説明する．

3.1. 定式化

まず，我々は有限的な有向グラフ$G=(V,E)$を定義する．V：ノード（データ）E：枝（プログラム）とする．データの流れを図 2 に示す．

$G=(V,E)$

$V=\{v_1,v_2,v_3,\cdots,v_n\}$

$E=\{(v_1,v_2),\cdots,(v_m,v_n)\}$

次に分かりやすさのため，例で説明する．

ある数式Gを下記のように定義している．

$G=(V,E)$

$V=\{v_1,v_2,v_3,v_4,v_5,v_6\}$

$$E=\begin{Bmatrix}(v_1,v_2),(v_2,v_3),(v_2,v_4),(v_3,v_5),\\(v_3,v_6),(v_4,v_5),(v_4,v_6)\end{Bmatrix}$$

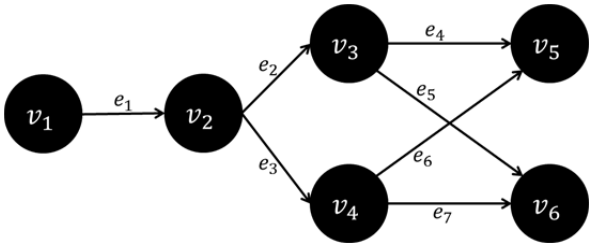

図 2 データの流れ

次に，比較のためにG'を定義する．G'はGに対応していくつのファイルを更新された状態である．以下のように表す．

$G'=(V',E')$

$V'=\{v_1',v_2',v_3',v_4',v_5',v_6'\}$

$$E'=\begin{Bmatrix}(v_1',v_2'),(v_2',v_3'),(v_2',v_4'),(v_3',v_5'),\\(v_3',v_6'),(v_4',v_5'),(v_4',v_6')\end{Bmatrix}$$

3.2. バージョン管理手法

最初に以下のような取り決めをする．

1. データの中身が異なっている場合は$v\neq v'$,同じなる場合$v=v'$と書く．
2. $E=\{(v_1,v_2),\cdots\cdots,(v_i,v_j)\},E=\{e_1,e_2,\cdots e_n\}$ とした時，それぞれのe_iに対して$\mathrm{e}=(v_j,v_k)$とする．
3. v_xの親ノード集合を$parent(v_x)$と書き」，$(v_y,v_x)\in E$となるようなv_yの集合である．

次は，定義された数式GとG'により，あるノードvが更新される条件に対して判断するアルゴリズムを示す．

Algorithm 1 need_update

Input $G \times G' \times v \times v'$

Output $TRUE, FALSE$

1. **if** $v \neq v'$ **then return** $TRUE$
2. **if** $parents(v) \neq \emptyset$
3. **for each** $w \in parents(v)$ **do**
4. **if**$(need\_update(G, G', w, w'))$
5. **then return** $TRUE$
6. **end for**
7. **for each** $e \in inedge(v)$
8. **if** $e \neq e'$ **then return** $TRUE$
9. **end for**
10. **return** $FALSE$

本アルゴリズムの入力はG, G', v, v'である．Gとvは定義された数式GとGのノードvで，G'とv'は比較するための数式G'とG'のノードv'である．出力はTRUEとFALSEである．TRUEはあるノードvを更新することで，FALSEはノードvを更新する必要がない．最初vの自分自身で判断する，$v \neq v'$の場合，vのあることを知る，vを更新する．$v = v'$の場合，vの化がないので，vの親ノードがあるかどうかを判断する，親ノードがない場合は，v更新しないで，親ノードがある場合は，vのすべて親ノードを探して，上記の循環を行い，vの親ノードの中に一つノード変化あれば，vを更新の必要がある，もしすべての親ノードを変化なかったら，枝の判断を行い．一つの枝を変化があればノードvを更新する，逆FALSEで出力する．

このアルゴリズムはSVNと連携に使うつもりである．例えば図１が示すように，予めにサーバのリポジトリとクライアントの作業スペースW1，W2にはそれぞれGがおいてある．もし作業スペースW1でGがG'になって，クライアントからリポジトリにコミットされたとする．すると，リポジトリではすべての$v \in V'$に対して自動的にこのアルゴリズムを実行する．もしあるv に対して$need_update(v)$ がTRUEならば，データを再計算しなければならない．再計算の順序など今後の研究で明確にする必要がある．

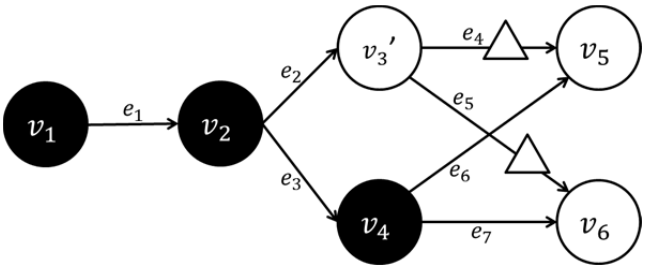

図 3　再計算の例（白い部分を再計算）

例えば、$v_3 = v_3'$の時，$need_update(v_5) = TRUE$なのでe_4とe_5を再計算する必要がある．

4. おわりに

本提案のグラフ表現には，プログラムの入力と出力が一つであるという制限などの問題があるため、さらなる改善が必要である．また，別のアルゴリズムを必要する．例えば，再計算の順序の特定などである．今後は，本バージョン管理手法に対してアルゴリズムを改善すると伴に，Subversion というバージョン管理システムを用いた実装方式を検討する．そして，実際に ALKAN プロジェクトの管理を実験し，効率について評価を行う予定である．

参考文献

[1] 服部 祐一, 井上 創造, 平川 剛, “行動情報共有システムにおける行動認識と可視化”, マルチメディア，分散，協調とモバイル(DICOMO)シンポジウム, pp.91-98, July 6, 2011.

[2] Yuichi HATTORI and Sozo INOUE, “A Large Scale Gathering System for Activity Data using Mobile Devices”, IPSJ Journal Vol.20 pp1-11,2011

[3] C. Michael Pilato, Ben Collins-Sussman, Brian W. Fitzpatrick , Version Control with Subversion, O’REILLY,2008

[4] The Apache Software Foundation, Apache Subversion, http://subversion.apache.org/

[5] Scott Chacon, Git - Fast Version Control, http://git-scm.com/

[6] the Mercurial community, Mercurial SCM, http://mercurial.selenic.com/

[7] Free Software Foundation, Inc., CVS-Open Source Version Control, http://www.nongnu.org/cvs/

問い合わせ先：
李林
九州工業大学大学院 工学府
先端機能システム工学専攻
E-mail: lilin19840702@gmail.com
Tel/Fax: 093-884-3410