社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

サービス指向リポジトリマイニングを効率化するキャッシュ機構の実装

坂元　康好†　　松本　真佑†　　中村　匡秀†

† 神戸大学 〒 657–8531 兵庫県神戸市灘区六甲台町 1–1
E-mail: †gen@ws.cs.kobe-u.ac.jp, ††{shinsuke,masa-n}@cs.kobe-u.ac.jp

あらまし　我々は先行研究において，リポジトリマイニングの各種技術をネットワーク上のサービスとして実現するフレームワーク SO-MSR（Service-Oriented Mining Software Repsitory）を提案した．さらに SO-MSR に従い，リポジトリの種類の違いやプログラミング言語の違いを気にすることなくメトリクスを算出する Web サービス，MetricsWebAPI を開発した．MetricsWebAPI の課題の一つに，メトリクス算出に係る処理時間が長く，応答時間の観点でユーサビリティが低いという点がある．本稿では，SO-MSR における各種 MSR サービスのインタラクション改善を目的として，処理結果再利用のための MSR キャッシュ機構と非同期 API の実現を目指す．メトリクス算出による処理結果や中間データは再度計算する必要がなく，キャッシュにより処理の効率化が見込める．また，非同期処理を導入することで，MSR 処理の並行・バッチ処理が可能となる．評価実験として，MSR 処理に対してキャッシュ機構を導入することにより，どの程度処理効率の改善が見込めるかを評価実験で確かめる．
キーワード　キャッシュ，非同期 API，マイニングソフトウェアリポジトリ，SO-MSR, MetricsWebAPI

Implementing Caching Mechanism to Improve Efficiency of Service-Oriented Mining Software Repository

Yasutaka SAKAMOTO†, Shinsuke MATSUMOTO†, and Masahide NAKAMURA†

† Kobe University, 1–1 Rokkodai, Nada, Kobe, Hyogo, 657–8531 Japan
E-mail: †gen@ws.cs.kobe-u.ac.jp, ††{shinsuke,masa-n}@cs.kobe-u.ac.jp

Abstract　We have proposed a framework called SO-MSR (Service-Oriented Mining Software Repository), which applied service-oriented architecture to MSR techniques. Following the SO-MSR, we have developed a web service, named MetricsWebAPI, for metrics calculation from a variety of software repositories and a variety source codes. One of the challenges of MetricsWebAPI is low usability in terms of response time for MSR processing requests. The goal of this paper is to improve service interaction for MSR services deployed in SO-MSR. In this paper, we apply a caching mechanism and asynchronous API to MSR services. In MSR processing, all of repository data are past logs and processed result can be reused to other MSR services if once MSR applied to the repository data. Asynchronous API realizes parallel and batch processing to MSR services. We have conducted a experimental evaluation to confirm the efficiency of applying a caching mechanism to MSR services.
Key words　Cache Asynchronous API, Mining Software Repository, SO-MSR, MetricsWebAPI

1. はじめに

Mining Software Repository（MSR）とは，ソフトウェアの開発履歴データが蓄えられたリポジトリをマイニングする行為や技術，あるいは研究分野のことを指す．MSR の実施により，定量的な根拠に基づいたプロセス改善や自己振り返りが実現できる．

我々は先行研究 [1] において，MSR の各種手法や技術をネットワーク上のサービスとして実現するフレームワーク，Service-Oriented MSR（SO-MSR）を提案している．SO-MSR はサービス指向アーキテクチャ（SOA）の考えに従い，MSR のあらゆる処理や技術，データなどをネットワーク上のサービスとして隠蔽することで，サービスの粗結合化と再利用性の向上を図る．SO-MSR を利用することにより，MSR に関する特別な知識のない MSR 実施者であっても，容易に MSR を実施しそのマイニング結果を取得することができる．また，他の MSR のサービスとの柔軟な組み合わせも可能であり，MSR の拡張や新たな MSR 手法のサービス化も実現可能である．

先行研究 [2] では，MSR の代表的な手法の一つであるソースコードメトリクスの計算に着目し，SO-MSR のフレームワークに従った Web サービス，MetricsWebAPI を開発した．MetricsWebAPI では，ソフトウェアリポジトリの違いや，計算対象となるソースコードのプログラミング言語の違いを意識することなく，単純なサービス API の呼び出しだけで必要なメトリクスを取得することができる．例えば，あるソースコードの行数を取得する場合，ソフトウェアリポジトリの登録 ID と対象ソースコードのパスを引数として URL 呼び出しの形式で呼び出すだけでよい．このサービスの呼び出しには SOAP や REST などの XML ベースのプロトコルで呼び出すことが可能であり，M2M（Machine to Machine）でのサービス連携が実現できる．現在，Web 上には様々な Web サービスが提供されており，それらとの連携することでさらなる拡張・応用が可能である．

MetricsWebAPI の一つの課題として，メトリクス算出に係る処理時間が長く，応答時間の観点でのユーザビリティが低いという点が挙げられる．例えば，ある 1 リビジョンのあるソースコードの行数を取得する場合，数秒程度で結果を取得できるが，複数リビジョンに渡って行数の変化を確認したい場合，数十秒から数分の処理時間が必要となる．対象のソースコード，リビジョン数が増加するほど，この応答時間が長くなる傾向にある．さらに SO-MSR では様々な MSR をサービス化し，サービス同士の組み合わせで新たな MSR の実現・拡張が可能である．しかしながら，サービス連携の際には連携するサービスの数に応じて応答時間が長くなりやすく [3]，単一サービスでの応答時間の遅さは SO-MSR の重要な要素であるサービス連携実現のハードルにもなり得る．MetricsWebAPI のユーザビリティ，および SO-MSR のサービス連携を実現するためには，SO-MSR フレームワーク全体に対し，ユーザの処理要求に対するインタラクションを改善できる枠組みが必須である．

本研究では，SO-MSR における各種 MSR サービスのインタラクション改善を目的として，処理結果再利用のためのキャッシュ機構と非同期 API の導入について検討する．MSR の入力となるリポジトリデータは基本的に過去のログデータであり，その処理結果や中間データは再度計算する必要がなく，キャッシュにより高い処理の効率化が見込める．また，非同期 API により応答時間の長い処理要求のスレッド占有を避け，ユーザによる MSR 処理の並行・バッチ作業が可能となる．評価実験として，MSR 処理に対してキャッシュ機構を導入することにより，どの程度処理効率の改善が見込めるかを評価実験で確かめる．

2. 準　　備

2.1 Service-Oriented MSR（SO-MSR）

SO-MSR とはサービス指向アーキテクチャ（SOA: Service-Oriented Architecture）の考えを取り入れた，MSR のためのフレームワークである．図 1 に SO-MSR のアーキテクチャを示す．ここでは版管理システムのリポジトリとして，CVS と SVN の 2 つが例示されている．このリポジトリ種類ごとのアクセス方法の違いは，RCS Service によって吸収される．RCS

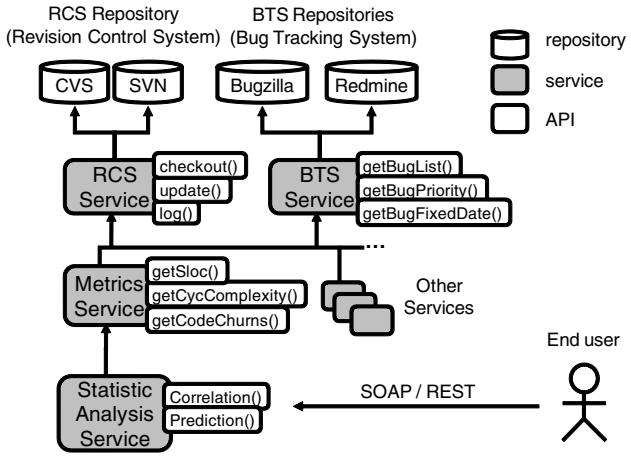

図 1　SO-MSR のアーキテクチャ

```
$ curl http://metrics.web.api/registerRepository?
repo=http://path.to.repo/
<id>1</id>

$ curl http://metrics.web.api/getSlocs?repo=1&src=
main.java
<getSlocs>
  <revision>
    <revid>113</revid>
    <date>2013-06-26</date>
    <author>gen</author>
    <sloc>358</sloc>
    <commitlog>main.javaの追加</commitlog>
  </revision>
  <revision>
    <revid>166</revid>
    <date>2013-07-27</date>
    <author>gen</author>
    <sloc>381</sloc>
    <commitlog>バグ#32の修正</commitlog>
  </revision>
  ...
</getSlocs>
```

図 2　MetricsWebAPI を用いて各リビジョンの行数を取得する際のコマンド

Service はリポジトリアクセスに関する操作を `checkout()` や `log()` などの API として公開している．バグ管理システムへのアクセスについても同様に BTS Service によってラップされており，`getBugList()` や `getBugFixedDate()` などの API を提供している．Metrics Service はメトリクス計算のためのサービスであり，RCS Service，BTS Service の両方のサービスを利用している．また統計解析のための Static Analysis Service は，この Metrics Service を利用してメトリクス値を取得し統計処理を施す．このように複数のサービスを組み合わせることで，柔軟に MSR サービスの拡張が可能となる．

2.2 MetricsWebAPI

我々は SO-MSR のフレームワークに従ったサービスの一つとして，MetricsWebAPI を開発している．MetricsWebAPI はソフトウェアメトリクス算出のための Web サービスであり，現在以下に示す 3 種類のソフトウェアメトリクスを計測できる．

- プロダクトメトリクス：総行数，CK，複雑度など
- プロセスメトリクス：変更行数，リビジョン回数など

- 開発者メトリクス：開発者数，開発者リストなど

MetricsWebAPI の利用例について説明する．cURL コマンド(注1)を用いて，MetricsWebAPI を利用した際の流れと結果を図 2 に示す．この図は，あるリポジトリ（http://path.to.repo/）に格納されているソースコード（main.java）について，main.java に関連する全リビジョンの総行数一覧を取得する場合の API 呼び出し例を表している．1 つ目の cURL コマンドでは，MetricsWebAPI へリポジトリを登録する `registerRepository()` を呼び出しており，2 つ目の cURL コマンドで，各リビジョンの行数一覧を取得する `getSlocs()` を呼び出している．このように単純なパラメタを組み合わせた URL 呼び出しだけで，ソースコードの言語の種類やリポジトリの種類を意識することなくメトリクスの算出結果を得ることができる．

また，サービスの呼び出しには，XML 形式のテキストベースプロトコル（SOAP/REST）が利用される．そのため，複数のサービス間での連携（マッシュアップ）が比較的容易に実現可能であり，BTS Service と組み合わせたバグ予測サービスなど様々なサービスへの拡張が期待できる．

2.3 課　　題

MetricsWebAPI の一つの課題として，メトリクス算出に係る処理時間が長く，応答時間の観点でのユーザビリティが低いという点が挙げられる．MetricsWebAPI は，ソフトウェアリポジトリからのデータ取得手段として，RCS Service（Revision Control System Service）を利用している．RCS Service 自体も SO-MSR の従う Web サービスであり，様々な種類のソフトウェアリポジトリ（CVS，SVN, Git 等）から，その種類の違いを意識することなく統一的にリポジトリを操作できる．現在，MetricsWebAPI と RCS Service のいずれも処理効率工夫の改善が一切施されていない．そのため，メトリクスの計算要求が発生する度に，リポジトリの Update 処理とメトリクス計算処理が行われており，特に複数リビジョンにまたがるマイニング処理を行う場合，数十分から最悪 1 時間以上の応答時間が必要である．また，HTTP ベースのプロトコルでサービスが利用されるため，SOAP/REST 等レスポンスのタイムアウトも発生しやすい．MetricsWebAPI，および MetricsViewer のユーザビリティを確保するためには，SO-MSR フレームワーク全体に対し，ユーザの処理要求に対するインタラクションを改善できる枠組みが必須である．

3. 提 案 手 法

前節で挙げた課題を達成するために，SO-MSR に対しキャッシュ機構と非同期 API の 2 つの考えを導入する．キャッシュ機構は，MSR 処理要求に対する結果や中間処理データを一時的に保存し，同じ処理要求が発生した場合に再度処理を行わずに一時保存された結果を返す仕組みである．非同期 API は，ユーザやクライアントからの処理要求に対し，MSR 処理を非同期に実行する枠組みである．具体的には，まず処理要求が発生した場合，結果取得のためのタスク ID のみを即時ユーザに返す．メトリクス計算などの本質的な MSR 処理は非同期に実行され，処理要求を行ったユーザは先ほど取得したタスク ID を用いて適宜，処理状況の確認と結果の取得を行う．

(注1)：コマンドラインから URL を呼び出すためのツール．

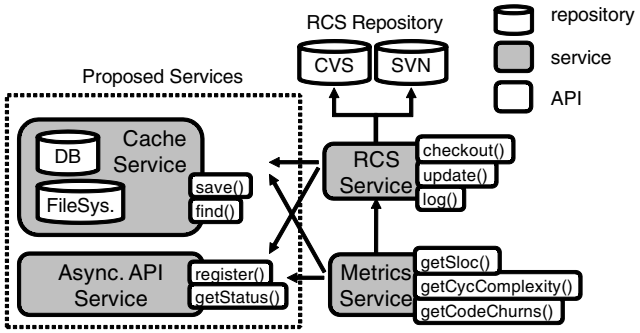

図 3　提案 Web サービスを加えた SO-MSR のアーキテクチャ

本論文では，SO-MSR との親和性や MSR サービス間の粗結合・再利用性を考慮し，これら 2 つの提案手法をそれぞれ Web サービスとして実現することを考える．手法全体の概略として，SO-MSR 上に提案 Web サービスを実現した際のアーキテクチャを図 3 に示す．図 3 は図 1（SO-MSR の全体アーキテクチャ）の左上部分（RCSService と MetricsService）のみを取り出したアーキテクチャに，2 つの提案サービス（破線部分）を加えた図である．

“CacheService” はサービス内にデータベースとディスクファイルシステムを内包し，MSR サービスが生成した多種多様なキャッシュデータを保持する．“AsynchronousAPIService” は処理タスクの登録 API と処理状況の取得 API を持ち，非同期での処理状況のモニタリングを実現する．両サービス共に，様々な MSR サービスからユーティリティサービスとして利用される．以降，4. 節でキャッシュ機構について，5. 節で非同期 API について説明する．

4. キャッシュ機構

4.1 概　　要

再利用性の高い MSR の処理結果や処理中間データを一時保持することで，再度 MSR 処理を行わずに処理効率を向上させる仕組みである．基本的なコンセプトはコンピューティングシステムにおける，ディスクキャッシュやブラウザのキャッシュと同じ仕組みである．本論文では，SO-MSR との親和性を考え，キャッシュ機構そのものの Web サービス化を図る．

4.2 種別の整理

キャッシュ機構の設計にあたって，まず MSR におけるキャッシュデータの種別に関する考察を行う．一般的な MSR 処理の流れを抽象化して考えた場合，以下の 3 つのステップから構成されると考えられる．

（1） リポジトリからの処理対象データの取り出し
（2） 処理対象データに対する MSR 処理の適用
（3） 処理結果の取得

各ステップについて発生するデータを考えると，MSR で発

表 1　キャッシュサービスの API 一覧

API 名	概要	引数	結果
registerCacheInfo	キャッシュデータのメタ情報の登録	図 4 の XML	なし
getCacheInfos	キャッシュデータのメタ情報一覧の取得	なし	図 4 の XML の一覧
save	キャッシュデータの登録	key, value	なし
find	キャッシュデータの取得	key	value

生するデータを以下の 3 種に大別できる．

（A） 対象の生データ

（B） 処理の中間データ

（C） 処理結果データ

（A）はソフトウェアリポジトリから取得したソースコードやログ，バグリポジトリから取得したバグ情報などの，未加工のデータのことを指す．(B) は MSR 処理の中間データのことであり，例えばメトリクス算出処理の場合，抽象構文木や空行・コメントの整形済みソースコードなどが含まれる．またバグ予測の場合のデータから構築された統計モデルや，ソーシャルマイニングの場合のコメントの自然言語済みデータなど様々な種類が含まれる．(C) は MSR 処理適用後の処理結果データであり，メトリクス算出処理の場合，メトリクスの数値データのことを指す．これには数値に限らず，テキストファイル，表形式のバイナリデータ，プロット済みの画像データなども含まれる．

上記の 3 つの各種データについて，いずれのタイプについてもキャッシュが MSR 処理効率の改善に有効であると考えられる．(A) の生データについては，様々な MSR 処理の入力となるため，複数の MSR サービスから横断的に利用可能なキャッシュデータであるといえる．(B) に関しても，MSR 処理結果自体のキャッシュは高い再利用性を持つ．(C) に関してはどの中間データをキャッシュするかは，そのデータの再利用の可能性を鑑みて考慮すべきである．例えば，抽象構文木は様々なメトリクス算出で利用可能なため，その再利用性は高くキャッシュすべきといえるが，予測のための統計モデルは様々な入力パラメタに強く依存しており，その再利用性は一概に高いとはいえない．どのタイプのキャッシュをどのタイミングで保存するかは，キャッシュ結果の再利用頻度や処理効率の改善効果を考慮に入れる必要がある．

4.3　参照局所性

MSR とキャッシュの親和性について，一般的なキャッシュシステムで用いられる 2 つの参照局所性の概念から検討する．

時間的局所性：参照データが「時間的」に近いうちに，再度参照される可能性が高いという特性を表す．MSR の場合，一度要求された MSR 処理は近い将来に（同じユーザによって）再度要求される可能性が高く，この特性を満たすといえる．例えば，MSR のクライアントアプリの更新（リフレッシュ）操作によって，同じ処理要求が発生するケースや，結果の確認のために再度同じ処理要求を行うといったケースが考えられる．

空間的局所性：参照データの「空間的」近傍のデータが，近い将来に参照される可能性が高いという特性を指す．MSR の場合，この「空間」に関しては，リビジョン方向の空間とソースコード周りの空間の 2 種類が考えられる．追加行数や開発者数などの複数リビジョンを入力とするプロセスメトリクス算出の場合，前者のリビジョン空間の空間的局所性が高い．また，パッケージメトリクス算出の場合，後者のソースコード空間の局所性が高い．すなわち，いずれの空間であっても，MSR キャッシュデータの空間的局所性は高いといえる．

```
<cacheInfo>
  <summary>ソースコードのASTに関するキャッシュ</summary>
  <key>MWA_AST_%revision%_%path%</key>
  <key_vars>
    <key_var>
      <name>revision</name>
      <format>%d</format>
      <description>特定のリビジョン</description>
    </key_var>
    <key_var>
      <name>path</name>
      <format>%s</format>
      <description>特定のファイルのフルパス</description>
    </key_var>
  </key_vars>
</cacheInfo>
```

図 4　キャッシュデータのメタ情報を表す XML ファイル

当然ながら MSR 処理の種類によって，いずれの局所性特性を満たしやすいかは異なる．例えば，前述の通りプロセスメトリクスは，リビジョン方向に関する空間的局所性が極めて高い．この性質を利用して，あるリビジョンの処理要求後に事前に別リビジョンのキャッシュを行うと行った，キャッシュのプリフェッチ（先読みキャッシング）機構を設けることで，さらなる効率化が可能となる．

4.4　保持期間

メモリキャッシュ等ではキャッシュ領域の制約が厳しく，キャッシュデータの保持期間の設定と破棄の工夫が必須である．MSR キャッシュでは，キャッシュ対象データが数値やファイルであり，キャッシュ領域としてデータベースやディスクファイルシステムが利用できる．近年ではディスクのストレージ領域が安価でかつ膨大であること，また分散ファイルシステムが容易に構築できる環境が整っている．このことから，ユーザからの明示的なクリア要求，あるいはキャッシュの生存期間が切れない限り，MSR キャッシュは破棄されないものとする．

4.5　キャッシュ機構の Web サービス化

MSR キャッシュ機構の Web サービス化（MSR キャッシュサービス）について検討する．MSR キャッシュデータの構造は一般的なキャッシュ機構と同様に，Key と Value の単純な組み合わせで構成されるものとする．設計した MSR キャッシュサービスの API 一覧を表 1 に，キャッシュデータのメタ情報を表す XML ファイルの例を図 4 に示す．

キャッシュデータの登録者は，まず `registerCacheInfo()`

表 2　非同期 API サービスの API 一覧

API 名	概要	引数	結果
register	処理タスクの登録	タスクのメタ情報	生成されたタスク ID
updateStatus	処理タスクの進行状況の更新	タスク ID，タスク進行状況	なし
getStatus	タスク進行状況の取得	タスク ID	タスクの進行状況
getResult	タスク処理結果の取得	タスク ID	タスクの処理結果

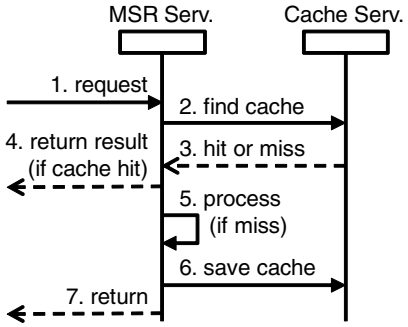

図 5　MSR キャッシュサービス利用のシーケンス図

API を利用してキャッシュデータのメタ情報を登録する．これは MSR サービスに横断的な MSR キャッシュデータの利用のための API である．登録されるメタ情報は図 4 の通りであり，この図の場合，MetricsWebAPI（MWA）が生成する抽象構文木（AST）をキャッシュの Key として登録する，という意味で “MWA_AST” を Key のプレフィックスとして指定している．さらに，どのリビジョンのどのソースコードの AST かを識別できるように，revision と path の両方を Key の変数として扱う．Key の変数は`<key_vars>`タグ内で説明されており，変数 revision の書式は “%d”，すなわち数値のみで構成されているということが示されている．キャッシュデータの保持は `save()` API で実行され，引数として上記の書式に従った Key 文字列と Value を指定する．Value の書式としては，テキストで表現できるキャッシュデータ（メトリクスの算出結果 “100” 等）はテキストで指定し，ファイルの形式で表現されるデータ（抽象構文木やプロット済み画像データなど）はそのままファイルとして登録できる．キャッシュデータの取得には `find()` API を利用する．Key と Value は `save()` API と同様である．

別の MSR サービス開発者が，他サービスで利用しているキャッシュデータを利用する場合，`getCacheInfos()` API を利用する．取得されるデータは `registerCacheInfo()` API で登録された，キャッシュデータのメタ情報の一覧である．サービス開発者はこの一覧を確認しながら，自サービスに利用可能なデータかどうかを判断する．

4.6　MSR キャッシュサービス利用の流れ

図 5 に MSR キャッシュサービス利用の流れを表すシーケンス図を示す．まず MSR 処理要求を受けた MSR サービスがキャッシュサービスに問い合わせを行う．キャッシュデータがヒットした場合，MSR サービスはそのまま値を返し，ヒットしなかった場合は MSR 処理を続行する．MSR 処理完了後にキャッシュサービスに Key と Value のペアの形でキャッシュを保存し，同時にユーザに処理結果を返す．

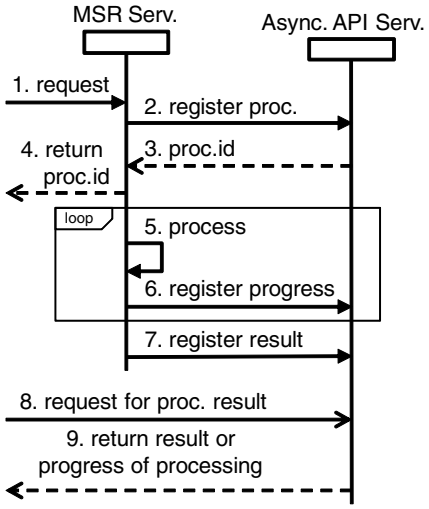

図 6　非同期 API サービス利用のシーケンス図

4.7　実　　装

MSR キャッシュサービスのプロトタイプシステムを Java 言語を使って開発した．現在，このプロトタイプシステムは 4.5 節で述べた API 一覧の内，`save()` と `find()` のみが利用できる．動作環境は Tomcat7 と Axis2 である．キャッシュ保持のためのファイルシステムとしては，ドキュメント指向 DB の一つである MongoDB と，NTFS フォーマットの一般的なディスクファイルシステムを用いた．プログラム総行数は約 200 行，開発工数は約 20 人日である．

5.　非同期 API

5.1　概　　要

実行時間の長い処理に対しては，処理スレッドの占有を避ける目的で非同期処理として実現することが一般的であり，SOMSR に対しても非同期 API の仕組みを取り入れる．サービス API を非同期で実現するという考えは facebook や Amazon Web Service の公開 API などでも取り入れられており [4] [5]，処理は非同期で行い，その結果を定期的に訪ねる仕組みが用いられる．これにより，本質的な処理時間そのものは改善されないが，ユーザは MSR 処理中であっても並行して別の MSR 処理を実施することができる．また非同期 API は HTTP リクエストのタイムアウト問題の回避にも繋がる．

5.2　非同期 API の Web サービス化

非同期 API を実現するサービス（非同期 API サービス）について説明する．表 2 に非同期 API サービスの API 一覧を，図 6 に非同期 API サービス利用時のシーケンス図を示す．

表 3　実験題材として用いたリポジトリの統計量

運用期間	2013 年 04 月〜09 月（現在）
総リビジョン数	223 回
最新リビジョンのチェックアウトサイズ	540MB

表 4　全リビジョン・全ソースコードのキャッシュ実験結果

キャッシュ取り込み時間	2 時間 25 分
キャッシュ総ディスク容量	89GB
キャッシュ総ファイル数	95 万個

まず，MSR 処理要求を受けた MSR サービスは非同期 API サービスに対し，タスクのメタ情報と共に `register()`API を呼び出してタスクを登録する．非同期 API サービスは新たなタスク ID を生成し返す．MSR サービスはユーザやクライアントの処理要求の返り値として，即時このタスク ID を返し，本質的な MSR 処理を非同期で実行する．MSR サービスは，MSR 処理最中に処理進行状況を，`updateStatus()` API を利用して非同期 API サービスに登録する．これによりユーザやクライアントはどのタイミングでも，`getStatus()` API を利用して MSR 処理の進行状況を把握することができる．処理を終えた MSR サービスは，処理結果を非同期 API サービスに登録し処理を終える．

6. 評価実験

6.1 実験手順

MSR 処理に対してキャッシュ機構を導入することで，どの程度処理効率の改善が見込めるかを実験により確かめる．実験題材としては，我々の研究室で利用しているソフトウェアリポジトリ（dev2013）を用いる．表 3 に，このリポジトリの統計量を示す．このリポジトリには 2013 年 4 月から現在までの研究室で開発したプロジェクトが保持されている．

実験の手順として，まず dev2013 の全てのリビジョン・全てのソースコードについて，4.2 節で述べた（A）のタイプのキャッシュデータ（リポジトリに保持された生ソースコード）をキャッシュする場合の実行時間とキャッシュサイズを計測する．次に，より現実的な MSR 処理題材として，特定のソースコード（ECAPlatform.java）を取り上げ，このソースコードの全リビジョンの行数の推移を取得する場合の処理時間を計測し，キャッシュありとなしの場合の速度を比較する．

6.2 実験結果

まず全リビジョン・全ソースコードのキャッシュ実験の結果を表 4 に示す．キャッシュ取り込み時間については，2 時間以上の時間が必要であった．特にリビジョン数 200 程度で運用期間が半年程度の小規模リポジトリであっても，このキャッシュ処理時間が必要であったことから，キャッシュ生成のタイミングに関する工夫が必要である．一つのアイデアとして，非同期 API サービスとキャッシュサービスを組み合わせることで，夜間の非同期バッチ型キャッシングが実現できる．これを利用して，MSR 処理要求が発生した場合にキャッシュを生成するだけでなく，定期的にキャッシュデータを事前に生成しておくといった工夫が可能である．

表 5　特定ソースコードを対象とした行数推移取得に関する実験結果

キャッシュありの処理時間	0.5 秒
キャッシュなしの処理時間	8.9 秒

次に，特定のソースコードを対象としたキャッシュありなし比較実験の結果を表 5 に示す．キャッシュを利用することでキャッシュなしの場合と比べて約 20 倍程度の高速化が見込めた．対象のソースコードは編集を行ったリビジョン数が 8 回だけであり，RCS Service の挙動としては 8 回のリポジトリ Update 操作が行われている．処理時間のほとんどはこのリポジトリ Update 操作が占めており，リビジョン数が多い（すなわち編集回数が多い）ソースコードほど処理時間が長く，キャッシュによる処理の効率化が見込めるといえる．

7. まとめ

本稿では，SO-MSR 上に配置された MSR サービスのインタラクション改善を目的として，MSR に対するキャッシュ機構と非同期 API の導入について検討した．さらに MSR キャッシュ機構のプロトタイプ Web サービスを開発し，その効果について実験を行った．実験によりキャッシュ機構によって，高い処理効率の向上とインタラクションの改善が見込めることを確認した．

今後の課題として，MSR キャッシュ機構と非同期 API のシステム開発，及び定量的な評価が必須である．また本稿の中では MSR キャッシュの生存期間については基本的に破棄しないというポリシーであったが，キャッシュデータは一方的に肥大化するため何かしらの破棄の仕組みが必須である．キャッシュのヒット率や再利用率を考慮した，MSR キャッシュの破棄や，MSR 処理後との参照局所性の特性を利用したキャッシュのプリフェッチなど様々な改善を試みる予定である．

謝辞

この研究の一部は，科学技術研究費（基盤研究 C 24500079, 基盤研究 B 23300009），および，積水ハウスの研究助成を受けて行われている．

文　献

[1] 柗本真佑，中村匡秀，“リポジトリマイニング技術共有のためのサービス指向フレームワーク,” ソフトウェア工学の基礎ワークショップ FOSE2011，vol.37，pp.231–236，2011.

[2] S. Matsumoto and M. Nakamura, “Service oriented framework for mining software repository,” Proceedings of the Joint Conference of the 21st International Workshop on Software Measurement (IWSM) and the 6th International Conference on Software Process and Product Measurement (Mensura), pp.13–19, 2011.

[3] D. Menasce, “Response-time analysis of composite web services,” Internet Computing, IEEE, vol.8, pp.90–92, 2004.

[4] “Amazon Web Service - Asynchronous Programming with the AWS SDK for Java”. http://aws.amazon.com/articles/5496117154196801.

[5] “facebook developers - Async API Requests”. https://developers.facebook.com/docs/reference/androidsdk/ayncrequest/.