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

信学技報
IEICE Technical Report
SS2006-68(2006-12)

D-CCFinder:超大規模ソースコード集合を対象とした分散処理型コードクローン検出・可視化システム

リビエリシモネ† 肥後 芳樹† 松下 誠† 井上 克郎†

† 大阪大学 大学院情報科学研究科 コンピュータサイエンス専攻 〒560-8531 豊中市待兼山町 1-3
E-mail: †{simone,y-higo,matusita,inoue}@ist.osaka-u.ac.jp

あらまし コンピュータハードウェアが安価になり，分散処理方式はソフトウェア分析のための現実的な選択肢の1つとして用いられるようになった．本稿では，超大規模ソースコードからコードクローンを検出するためのシステムD-CCFinderについて述べる．D-CCFinderは80台のコンピュータを用いた分散処理型コードクローン検出システムであり，検出されたコードクローン情報は散布図などを用いて可視化される．D-CCFinderは約4億行のソースコードから2日余りでコードクローン情報を収集し，頻出するコードを容易に特定することができた．このような超大規模ソフトウェアからのコードクローン検出は，大量のソフトウェア間でのコードクローンの状態を把握すると共に，著作権違反のコード特定などにも応用することができる．

キーワード 分散処理，メガソフトウェアエンジニアリング，コードクローン，再利用

A Very-Large Scale Code-Clone Analysis and Visualization

Simone LIVIERI†, Yoshiki HIGO†, Matsushita MAKOTO†, and Katsuro INOUE†

† Graduate School of Information Science and Technology, Osaka University
1-3, Machikaneyama-cho, Toyonaka, Osaka, 560-8531, Japan
E-mail: †{simone,y-higo,matusita,inoue}@ist.osaka-u.ac.jp

Abstract The increasing performance-price ratio of computer hardware makes possible to explore a distributed approach at code clone analysis. This paper presents D-CCFinder, a distributed approach at large-scale code clone analysis. D-CCFinder has been implemented with 80 PC workstations in our student laboratory, and a vast collection of open source software with about 400 million lines in total has been analyzed with it in about 2 days. The result has been visualized as a scatter plot, which showed the presence of frequently usedcode as easy recognizable patterns. Also, D-CCFinder has been used to analyze a single software system against the whole collection in order to explore the presence of code imported from open source software.

Key words Distributed computing, Mega software engineering, Code clone, Reuse

1. はじめに

近年，ソフトウェア分析手法としてコードクローン検出が注目を集めている[7],[9]．コードクローン検出を行うことにより，単純な重複部分の調査だけではなく，ソフトウェアの進化を追うことができる[8][10]．

多くのオープンソースソフトウェアが開発されており，その一部は他のソフトウェアでも用いられている[2]．著者らはこれらのソフトウェアに対してコードクローン検出を行い，それらの間の関係を明らかにしようと試みている．

しかし，大量のソフトウェアからコードクローン検出を行うためには，既存の検出システムにおけるスケーラビリティの問題を改善しなければならない．多くのコードクローン検出技術がこれまでに提案されており，字句単位の検出手法を実装したツールCCFinder[6]が他の手法に比べ，スケーラビリティが高いものの1つとして知られている[3]．

本稿では，コードクローン検出対象として，オープンソースオペレーティングシステムFreeBSD用のソフトウェア集合Portsシステムに含まれている，C言語で記述された約4億行のソースコード（以降，オープンソースターゲット）を用いた．この規模は，CCFinderの検出可能である限界を遥かに超えており，単一のコンピュータ上で検出を行うことは難しい．そこで，分散処理方式を用いて，超大規模ソースコードを小さく分割し，各コンピュータに入力として与える．各コンピュータ上でCCFinderを実行し，割り当てられたソースファイルに含まれるコードクローンを検出する．このように対象を細かく分割することにより，既存のCCFinderで検出を行うことができる．仮にこの分割した全てのタスクを単一コンピュータ上で実行した場合は，約40日を要すると予測され(注1)，現実的ではない．

(注1)：CPU:Xeon 2.8GHz, Memory: 4GB を用いた場合

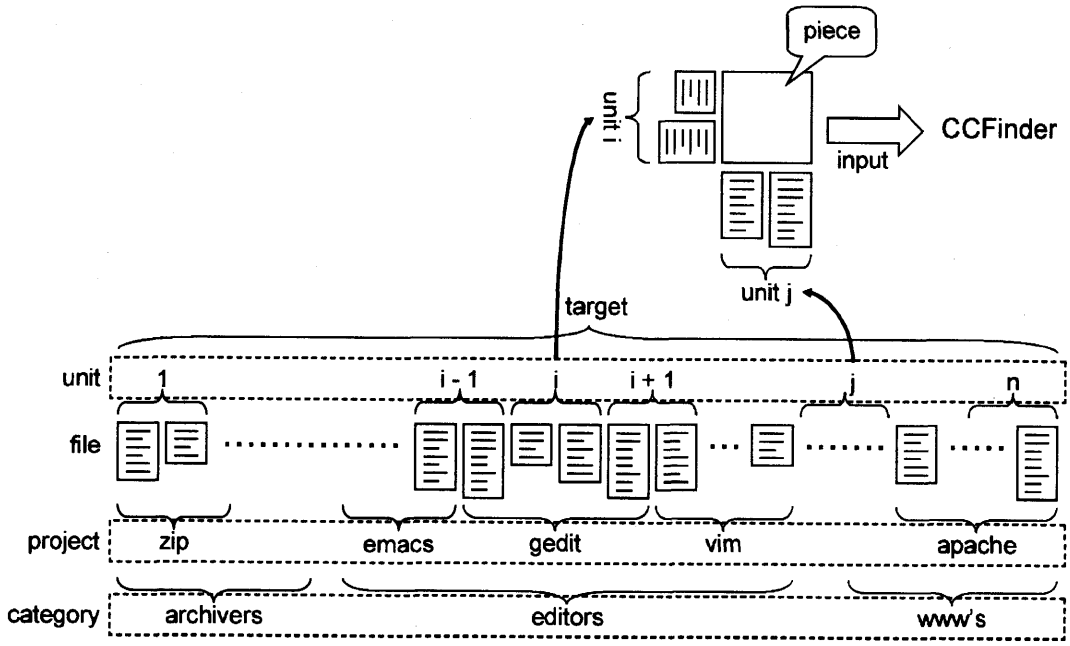


図 1　プロジェクト，カテゴリ，ターゲット，ユニット，ピース間の関係

しかし，分散処理で行うことにより，高速にコードクローンを検出できる．

提案手法を分散処理型アプリケーション Distributed-CCFinder（以降，D-CCFinder）として実装した．D-CCFinder を大学の演出室のコンピュータ 80 台上で実行し，オープンソースターゲット内に含まれるコードクローンを検出したところ，約 2 日で検出を完了することができた．

本提案の成果は以下の通りである．

- 分散処理方式を用いた，超大規模ソースコード用コードクローン検出手法を提案する．
- オープンソースターゲットに含まれるソフトウェア間で共有されているコードクローンの全体像を明らかにする．このような分析は著者らが知る限り過去に行われていない．
- 著作権違反の検出を目的として D-CCinder を用いる．この場合は，対象の単一ソフトウェアとオープンソースターゲット間で共有されているコードクローンを検出することになる．

以降，2. 節では D-CCFinder の計算モデルを定義し，3. 節ではその実装について述べる．4. 節ではオープンソースターゲットへの適用について述べ，5. 節ではその結果に対する考察を行う．最後に 6. 節では，まとめと今後の課題について述べる．

2. 超大規模ソースコードを対象としたコードクローン検出手法

2.1 検出対象

本稿でのコードクローン検出対象は，オープンソースオペレーティングシステム FreeBSD 用のソフトウェア集合 Ports システムに含まれているソースファイル（オープンソースターゲット）であり，各ソースファイルは 1 つの**プロジェクト**に属している．全てのプロジェクトは，zip, emacs, apache, windowmaker など，一意に特定可能な名前を持っている．また，本稿では検出対象を C 言語で記述されたソースコードに限定しているが，Java や COBOL などの CCFinder 自体が扱えるプログラミング言語であれば，同様に適用可能である．

共通の特徴を持ったプロジェクトは同じ**カテゴリ**に所属している．たとえば，emacs や vim，gedit などは editors カテゴリに所属している．

ユニットは，全ファイル集合を分析者が予め決めたサイズ以下で分割した要素であり，1 つのユニットに含まれるソースファイルは単一プロジェクトからなる場合と複数プロジェクトからなる場合がある．任意の 2 つのユニットで指定されるファイル集合間の対応を**ピース**といい，これが各コンピュータ上で実行される CCFinder への入力となる．図 1 はプロジェクト，カテゴリ，ユニット，ピース間の関係を表している．

2.2 検出結果の表示

コードクローンには，コメントを除く部分が全く同一の *exact* クローンと，変数名や関数名などのユーザ定義名が異なる *parameterized* クローンの二種類があるが [1]，本稿ではこの両方を検出する．

検出対象が非常に大規模であるため，大量のコードクローンが検出されることが予測される．このことから，コードクローンの全体像を把握するためには検出結果の抽象化を行う必要がある．提案手法では，コードクローン検出後に，ファイル，プロジェクト，カテゴリの各レベルで抽象化を行う．例えば，各ファイル間，各プロジェクト間，カテゴリ間のコードクローンカバレッジや，単にコードクローンを共有しているかどうかといった抽象化も行う．

2.3 計算モデル

検出対象規模が非常に大きいため，一度に対象全体からコードクローン検出を行うことは不可能である．このため，検出対象を小さなピースに分割し，ピース単位で CCFinder を実行することにより，コードクローンを検出する．

図 2 は D-CCFinder の計算モデルを表している．検出対象の規模は nu とする（n は分割数であり，u はユニットのサイズを表している）．このとき，任意のピースは (i,j) で表すことができる（ただし，$1 \leqq i,j \leqq n$）．ピース (i,j) に含まれるコードクローンはピース (j,i) に含まれるコードクローンと同様であるため，後者については検出を行わない．これにより，CCFinder を用いてコードクローンを検出しなければならないピースの数は $n(n+1)/2$ となる．

D-CCFinder は，既存の CCFinder を複数のコンピュータで実行することにより，コードクローン検出を行う．各ピースの演算（コードクローン検出）は他のピースの演算結果に全く依存しないため，タスクの割り当て処理は単純に行える．コードク

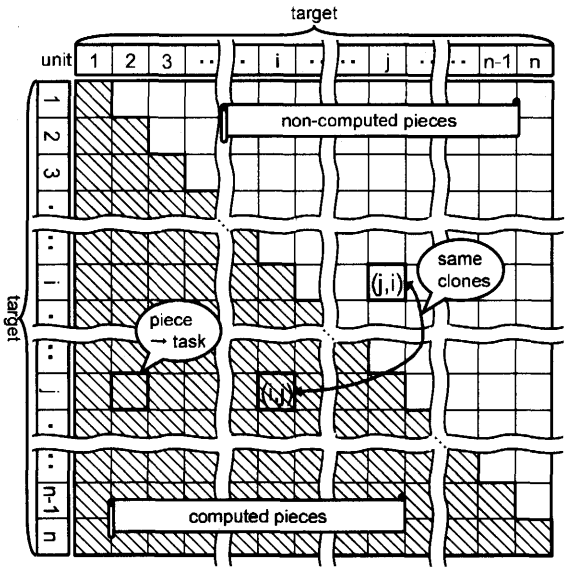


図 2　D-CCFinder の計算モデル

ローンが未検出のピースを，アイドル状態のコンピュータに割り当て，検出結果を回収するだけでよい．

3. D-CCFinder

D-CCFinder はマスタースレーブ型のシステムであり，各スレーブマシン上で CCFinder が実行される．マスターは，スレーブの実行状態を監視し，タスクを割り当てる．マスター・スレーブ間の通信は，Java RMI を用いて行われる．表 1 はマスターと各スレーブマシンの性能を表している．マスター・スレーブ間は 100Mbps のネットワークで結ばれている．検出対象ソースファイルと検出結果は全てのマシンがアクセス可能なファイルシステム上に存在し，各マシンは NFS 経由でアクセスする．D-CCFinder は大学の演習室のコンピュータ 80 台を用いて実装されており，1 台がマスター，残りの 79 台がスレーブである．

図 3 に示すように，D-CCFinder には対象ソースファイルに前処理を行うユーテリティ，検出結果を集約するプログラム，および散布図などを生成するジェネレータが統合されている．

Indexer　検出対象ソースファイルを走査し，ファイルサイズ，行数，プロジェクト名，カテゴリ名などの情報を収集する．また，ユニットの境界を決定する．

マスターノード　スレーブノード上の CCFinder の実行状態を監視する．アイドル状態のスレーブノードを発見した場合は，ユニット境界情報から，CCFinder の入力ファイル（そのユニットに含まれているソースファイルのパスのリスト）を生成し，スレーブノードにタスクを割り当てる．もし割り当てが失敗した場合は，そのタスクは他のスレーブに割り当てられる．

スレーブノード　マスターノードから与えられた入力ファイルを用いてコードクローン検出処理を行う．検出対象ファイルはスレーブノードのローカルファイルシステムのコピーされ，その後コードクローン検出が行われる．検出処理が終わった後もローカルファイルシステム上のコピーは削除されず，次回以降の検出処理のキャッシュとして利用される．

表 1　マスター・スレーブノードの性能

CPU	Pentium IV 3GHz
Memory	1 GBytes
OS	FreeBSD 5.3-STABLE
Local storage	40 ~ 50 GBytes

Clone Coverage Analyzer　D-CCFinder の出力から，ファイル，プロジェクト，およびカテゴリレベルのコードクローンカバレッジを算出する．

Image Generator　Clone Coverage Analyzer が生成した定量データから，散布図やヒートマップ等を生成する．

4. 実　　験

本稿での実験対象は，オープンソースオペレーティングシステム FreeBSD(http://www.freebsd.org/) 用のソフトウェアの集合である Ports システムに含まれているソースファイル（オープンソースターゲット）である．表 2 に対象ソースファイルの規模を示す．Ports システムに含まれているソフトウェアは管理者によりカテゴリ分けされている．

オープンソースターゲットは同じプロジェクトの複数のバージョンを含んでいる場合がある．例えば Apache web server の場合は，1.3, 2.0, 2.1, 2.2 の 4 つのバージョンが含まれている．これは，古いバージョンを必要としているユーザやシステムとの下位互換性を保つためである．このように，複数のバージョンが含まれる場合は，それらの間で非常に多くのコードクローンが検出されることが予測される．

D-CCFinder とオープンソースターゲットを用いて 2 つの実験を行った．初めの実験は，オープンソースターゲット全体のコードクローンの状態を調査した．次の実験では，著者らの研究室で過去に開発したシステムとオープンソースターゲット間のコードクローンについて調査した．

コードクローンの状態を定量的に表すために，2 つのメトリクス $Coverage(M_0, M_1)$ と $Coverage_{M_1}(M_0)$ を定義する．これらは次式で定義される．

$$Coverage(M_0, M_1) = \frac{LOC(C(M_0, M_1)) + LOC(C(M_1, M_0))}{LOC(M_0) + LOC(M_1)}$$

$$Coverage_{M_1}(M_0) = \frac{LOC(C(M_0, M_1))}{LOC(M_0)}$$

ただし:
M_0, M_1: ファイル，プロジェクト，またはカテゴリ，
$C(M_0, M_1)$: M_0 の中で，M_1 とコードクローンである部分，
$LOC(x)$: x の行数．

$Coverage(M_0, M_1)$ は M_0 と M_1 がどの程度コードクローンを共有しているのかを表し，$Coverage_{M_1}(M_0)$ は，M_0 が M_1 とどの程度コードクローンを共有しているのかを表す．

4.1 オープンソース全体の調査

最小一致トークン数(注2)を 50，ユニットサイズを 15MBytes に設定し，D-CCFinder を実行した．実行されたタスクの総数は，269,745 個である．図 4 は全体の散布図を表している．この散布図では，1 ドットあたり 200x200 ファイルを表している．200x200 ファイル間で 1 つでもコードクローンが存在する場合は，点を描画している．ファイルレベルでの $Coverage(M_0, M_1)$ の平均は 4%であり，もしこのような縮退を行っていない場合は，これより遥かに点の少ない散布図になると思われる．

図 4 の特徴的な部分を枠で囲んでいる．これらの部分に対して，より詳細に調査を行った．

表 2　オープンソースターゲットのサイズ

カテゴリ数	45
プロジェクト数	6,658
.c ファイル数	754,552
総行数	403,625,067
総容量	10.8 GBytes

(注2)：最小一致トークン数とは，CCFinder がコードクローンを検出する際に用いる閾値．CCFinder はこの長さ以上のコードクローンを検出する．

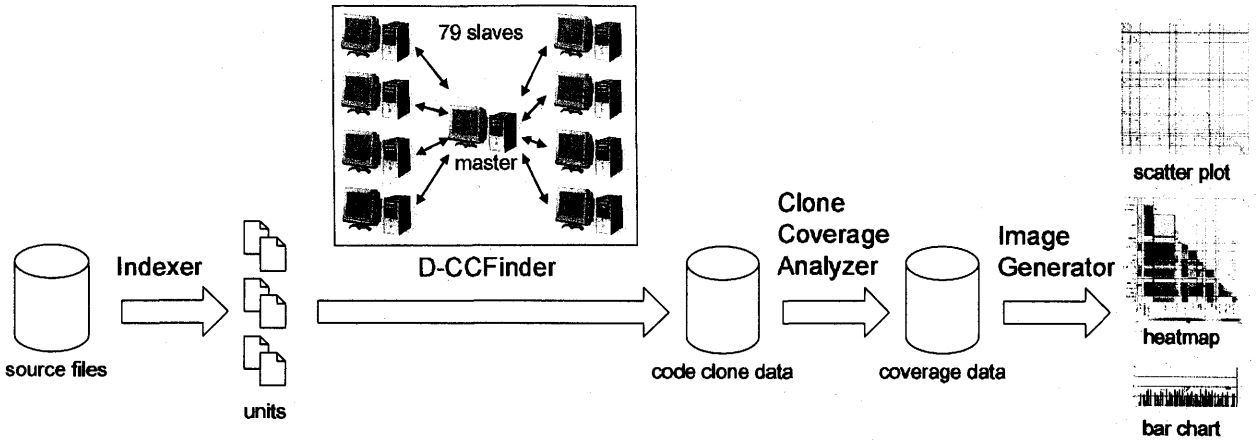


図 3　D-CCFinder の処理の流れ

A　この部分のコードクローンは php4 と php5 のソースコードが流用されていることを表している．図から読み取れるように，様々なカテゴリのプロジェクトに流用されている．
B　この部分には X11 関係の 4 つのカテゴリが存在しており，それらの間で多くのコードクローンが検出された．その多くは X Window System の中心的な処理を行っている部分からのコピーであった．
C　imake は make に代表されるビルドツールの一種であり，X Window System の一部となっているソフトウェアである．このソフトウェア自体はカテゴリ devel に属しているが，X11 関係のソフトウェアの多くがこのコピーを所持していた．
D　カテゴリ devel の大部分で一様なパターンが現われていた．これはソフトウェア binutils のコードの流用によるものである．binutils はリンカやアセンブラなどその他オブジェクトファイルやアーカイブを扱うためのソフトウェアツール群であり，カテゴリ devel に属するソフトウェアがこのツールを流用しているのは納得できる結果である．
E　カテゴリ audio 内に存在するマルチメディアフレームワーク gstreamer とその複数のプラグインが多数の同一ファイルを所持していた．

上記のコードクローンを所有しているファイルに対してメトリクス $Coverage(M_0, M_1)$ を計測したところ，ほとんどが 100%であった．つまり，これらのファイルはある一部ではなく，ファイル全体がコードクローンになっていることを表している．すでに述べたように図 4 の 1 ドットは 200x200 ファイルを表しているため，特定の 2 つのプロジェクト間でのみコードクローンになっている部分を発見することは難しい．

図 5 はカテゴリレベルでのメトリクス $Coverage(M_0, M_1)$ のヒートマップを表している．この図から，主対角線上，つまりカテゴリ内のカバレッジがカテゴリ間のカバレッジに比べ高いことがわかる．異なるカテゴリ間の場合は，$Coverage(M_0, M_1)$ の値が 25%を超えている箇所は少ない．次に，図 5 の特徴的な部分にどのようなコードクローンが存在していたのかを示す．
F　カテゴリ database の値が 41%と非常に高かった．これには 2 つの理由がある．1 つめの理由はこのカテゴリに属するいくつかのソフトウェアは，複数バージョンのソースコードが存在したこと，2 つめの理由は，ruby や php など異なるプログラミング言語向けにデータベースドライバが提供されている点であった．前者の理由により，ファイルの一部分のコードクローンが多く存在し，後者の理由により，ファイル全体のコードクローンが多く存在した．
G　カテゴリ devel の値が 38%であった．このカテゴリにはプロジェクト binutils や gcc の複数のバージョンが含まれており，

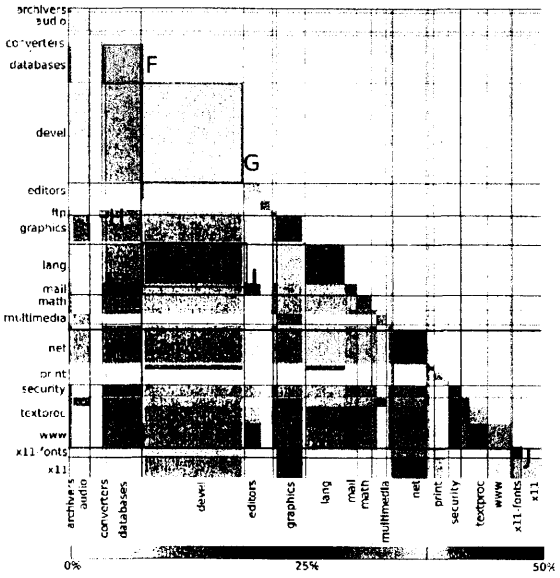


図 5　オープンソースターゲット全体のカテゴリレベルでのヒートマップ

$Coverage(M_0, M_1)$ の値を押し上げていた．
H　カテゴリ ftp と converters の間の値が 37%であった．これらのカテゴリに含まれる複数のソフトウェアがプロジェクト php4 と php5 のソースコードを流用しているため，$Coverage(M_0, M_1)$ の値が高くなっていた．
I　カテゴリ lang と devel の間の値が 28%であった．これはカテゴリ devel 内に複数バージョンのプロジェクト gcc が存在しているためであり，このコードはカテゴリ lang に含まれるプロジェクトでも流用されていた．
J　カテゴリ x11-fonts の値が 46%と最も高い数値であった．このカテゴリに属しているソフトウェアは少数であり，X Window System からのコードの流用が多く（7 箇所）行われていたため，このような高い数値になっていた．

4.2　SPARS-J とオープンソース間の調査

近年，オープンソースソフトウェアはめざましい進歩を遂げている一方で，既存システムのコードを再利用する際の著作権違反が問題になっている．著者らは，D-CCFinder を用いたコードクローン検出が，著作権違反問題に適応できると考えている．

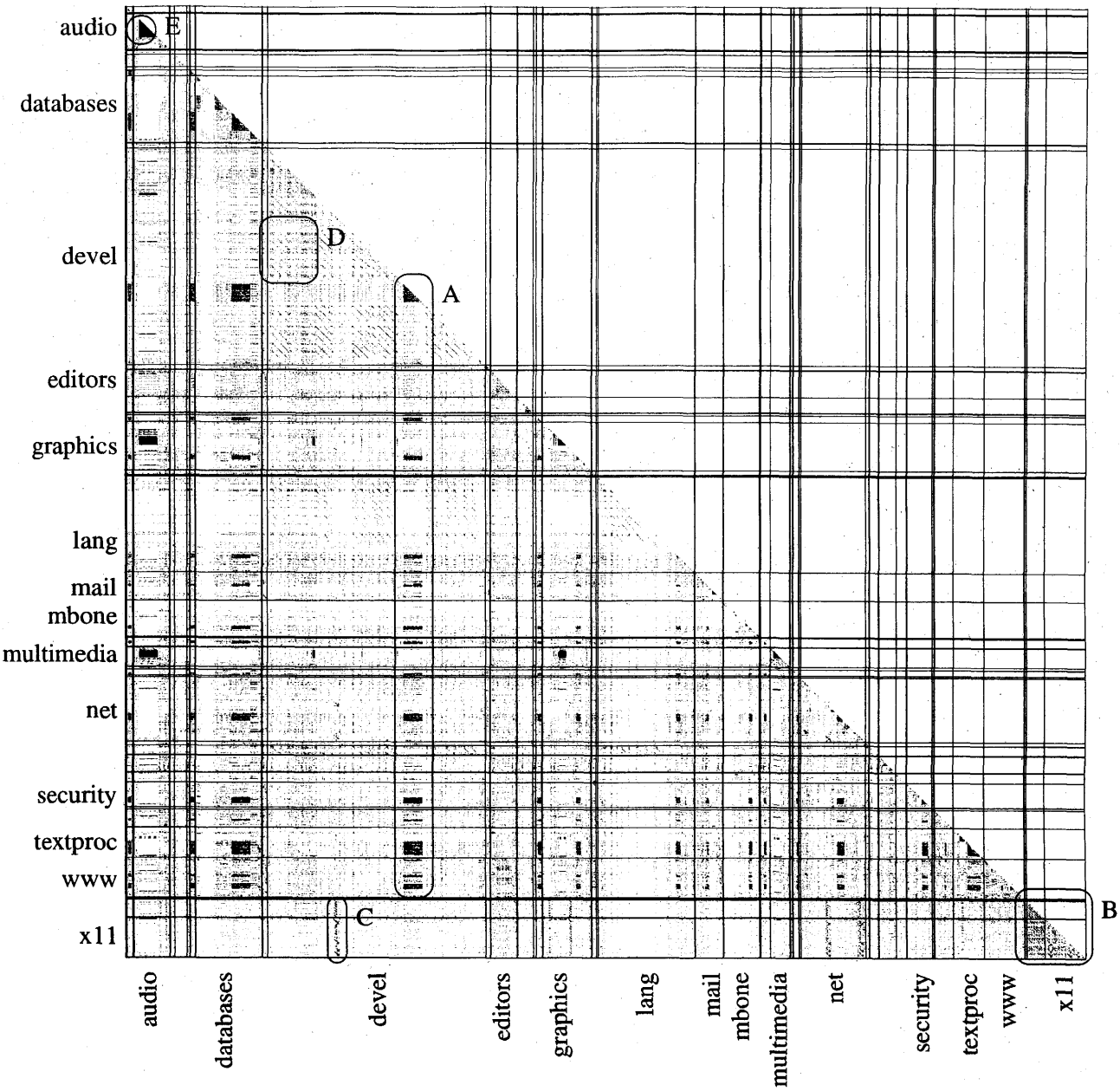


図 4　オープンソースターゲット全体の散布図

本実験では，SPARS-J(注3)とオープンソースターゲット間のコードクローン検出を行った．SPARS-J は C 言語で記述されており，サイズは約 47,000 行である．本実験では,SPARS-J がどのようにオープンソースターゲットとコードクローンを共有しているのかを調査する．この実験では，1 つのピースは 15MBytes のオープンソースターゲットのユニットと SPARS-J のソースコードからなり，合計 734 個のタスクが実行される．

図 6(a) は SPARS-J とオープンソースターゲット間でのプロジェクトレベルの $Coverage_{M_1}(M_0)$ の様子を表している．グラフの縦軸が $Coverage_{M_1}(M_0)$ の値を表している．

調査の結果，SPARS-J がコードクローンを共有している大部分は，ファイル getopt.c であることがわかった．このファイルは，コマンドラインオプションをパースする機能を実装していた．オープンソースターゲットに含まれる多くのプロジェクトが同名のファイルを同様の目的で用いていた．

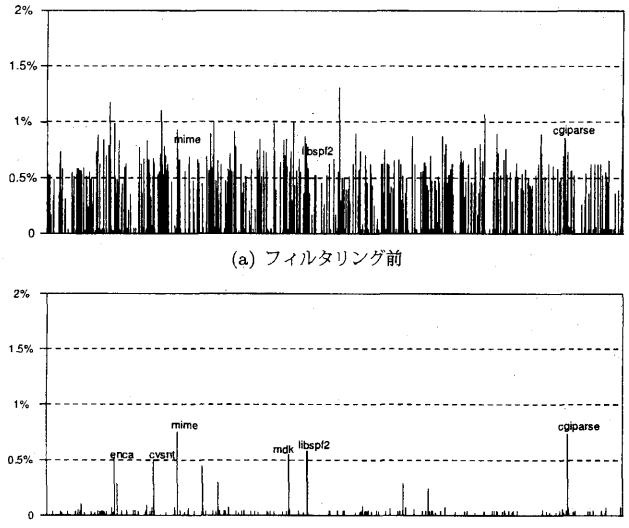
次に，ファイル名が getopt.c であるファイルを図 6(a) から取り除き，$Coverage_{M_1(M_0)}$ を算出した．図 6(b) はその結果を表している．この図から，SPARS-J は他のオープンソースソフトウェアとほとんどコードクローンになっていないことがわかる．しかし，少数ではあるが，$Coverage_1(M_0)$ の値が 0.5%以上のソフトウェアがいくつか存在している．これらのプロジェクトを調査した結果，CGI リクエストの処理部分がコードクローンになっていることがわかった．

5. 考　　察

5.1 オープンソース全体の調査

80 台のコンピュータ上で D-CCFinder を実行した結果，約 51 時間で検出を完了することができた．また，Clone Coverage Analyzer と Image Generator は研究室内の高性能ワークステー

(注3)：SPARS-J は著者らの研究室で過去に開発した Java 言語用ソースコード検索システムである [5].



(a) フィルタリング前

(b) フィルタリング後

図 6 SPARS-J とオープンソースターゲット間の重複度

ション(注4)上で実行した．

理論上は，80 台のコンピュータを用いることにより 12 時間で検出が完了するはずであるが，実際にはネットワークトラフィックやマスター・スレーブ間の同期，CCFinder の出力の後処理などのため 51 時間を要した．この検出速度は単一のコンピュータ上で行った場合の 20 倍である．現在，各コンピュータは 100BASE のスイッチで接続されており，ギガビットスイッチなど，高速なネットワーク環境を導入することにより，検出速度の向上が見込まれる．

D-CCFinder をクラスタ計算機やグリッド計算機 [4] で実現することも可能であろう．クラスタ計算機で実現する場合は，ネットワークの遅延等が減少し，全体の効率の向上が期待される．グリッド計算機では，大量の入出力データの効率的な分配・回収方法を実現する必要があろう．

また，散布図をより正確に生成する必要がある．現在の生成方法では，速度とサイズを重視しているため，精度が悪い（1 ドットが 200x200 ファイルを表している）．このため，対象全体の状態を大まかに把握することができるが，小さなプロジェクト間のコードクローンの状態を把握することはできない．

現在の実装では，Clone Coverage Analyzer は 1 台のワークステーションで実行されるため，処理に長い時間を必要とする．しかし，コードクローン検出と同様に，スレーブノードに分割して割り当てることにより，速度の向上を図ることができる．

5.2 SPARS-J とオープンソース間の調査

コードクローン検出は約 40 分で完了した．図 6(a) と図 6(b) が表しているように，SPARS-J はオープンソースターゲットとほとんどクローンを共有していないことがわかった．また一部のコードクローンを共有しているファイルを容易に突き止め，それがどのような機能を実装しているのかを知ることができた．このことから，D-CCFinder はソフトウェアの著作権違反調査にも用いることができる．

6. まとめ

本稿では，超大規模ソースコード集合から効率よくコードクローンを検出する手法を提案した．提案手法を分散システム D-CCFinder として実装し，オープンソースオペレーティングシステム FreeBSD 用のソフトウェア集合である Ports システムに含まれるソースファイルに対して適用した．約 4 億行の C 言語で記述されたソースコードから 51 時間でコードクローン検出を完了し，散布図やヒートマップを用いてコードクローン共有状態の全体像を把握することができた．また，著者らの研究室で過去に開発したシステム SPARS-J との間でコードクローンを検出した．検出結果は可視化され，容易にどの機能がオープンソースとコードクローンであるかを特定することができた．

本稿では，ソフトウェアエンジニアリングを分散環境で行うことの有用性を示すことも目的としている．D-CCFinder は超大規模ソースコードからコードクローン検出を行うための，単純で実用的なシステムであり，既存のネットワーク環境を用いて実装されている．

D-CCFinder は分散環境を用いたコードクローン検出のプロトタイプシステムであり，多くの改良しなければならない部分がある．今後は，CCFinder のではなく，fingerprint 技術を用いてコードクローン検出を行うことを予定しており，よりパフォーマンスの向上が見込まれる．

謝辞 大学の演習室を利用するにあたり協力していただいた大阪大学 大学院基礎工学研究科の田島滋人氏及び情報科学研究科の小泉文弘氏に感謝する．本研究は一部，文部科学省リーディングプロジェクト「e-Society 基盤ソフトウェアの総合開発」の委託を受けて行われた．また，科研費 基盤研究 (A)(No.17200001) 及び萌芽研究 (No.18650006) の助成を得た．

(注4)：CPU:Xeon 2.8GHz, Memory: 4GB

文献

[1] S. Bellon and R. Koschke. A comparison of automatic techniques for the detection of duplicated code. Technical report, Institute for Software Technology, University of Stuttgart, 2003.

[2] A. W. Brown and G. Booch. Reusing open-source software and practices: The impact of open-source on commercial vendors. In *Proc. of the 7th International Conference on Software Reuse*, Vol. 2319 of *Lecture Notes in Computer Science*, pp. 123–136, Austin, Texas, April 2002. Springer.

[3] E. Burd and J. Bailey. Evaluating clone detection tools for use during preventative maintenance. In *Proc. of the 2nd IEEE International Workshop on Source Code Analysis and Manipulation*, pp. 36–43, Oct 2002.

[4] I. Foster. What is the grid? a three point checklist, July 2002. available at `http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf`.

[5] K. Inoue, R. Yokomori, T. Yamamoto, M. Matusita, and S. Kusumoto. Ranking significance of software components based on relations. *IEEE Transaction on Software Engineering*, 31(3):213–225, April 2005.

[6] T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: A multi-linguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering*, 28(7):654–670, July 2002.

[7] C. Kapser and M. Godfrey. Improved tool support for the investigation of duplication in software. In *Proc. of the 21st International Conference on Software Maintenance*, pp. 25–30, Budapest, Hungary, September 2005.

[8] M. Kim, V. Sazawal, D. Notkin, and G. C. Murphy. An empirical study of code clone genealogies. In *Proc. of the 10th European software engineering conference*, pp. 187–196, Lisbon, Portugal, 2005.

[9] D. C. Rajapakse and S. Jarzabek. An investigation of cloning in web applications. In *Proc. of the 5th International Conference on Web Engineering (ICWE 2005)*, Lecture Notes in Computer Science, pp. 252–262, Sydney, Australia, 2005. Springer.

[10] T. Yamamoto, M. Matsushita, T. Kamiya, and K. Inoue. Measuring similarity of large software systems based on source code correspondence. In *Proc. of the 6th International PROFES (Product Focused Software Process Improvement) Conference*, pp. 530–544, Oulu, Finland, 2005.