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

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 井上研究室

リビエリ シモネ (simone@ist.osaka-u.ac.jp)
肥後 芳樹 (y-higo@ist.osaka-u.ac.jp)
松下 誠 (matusita@ist.osaka-u.ac.jp)
井上 克郎 (inoue@ist.osaka-u.ac.jp)

モチベーション

Software Engineering Laboratory

- これまでは単一のソフトウェアからのコードクローン検出が中心
  - 漏れの無いバグ修正・機能追加，水増しコードの指摘，純粋開発規模の把握，リファクタリング…

- 現在，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再利用されている
  - ソフトウェア間での類似度の調査により，再利用の把握ができる
  - 著作権違反コードの検出にも応用できる

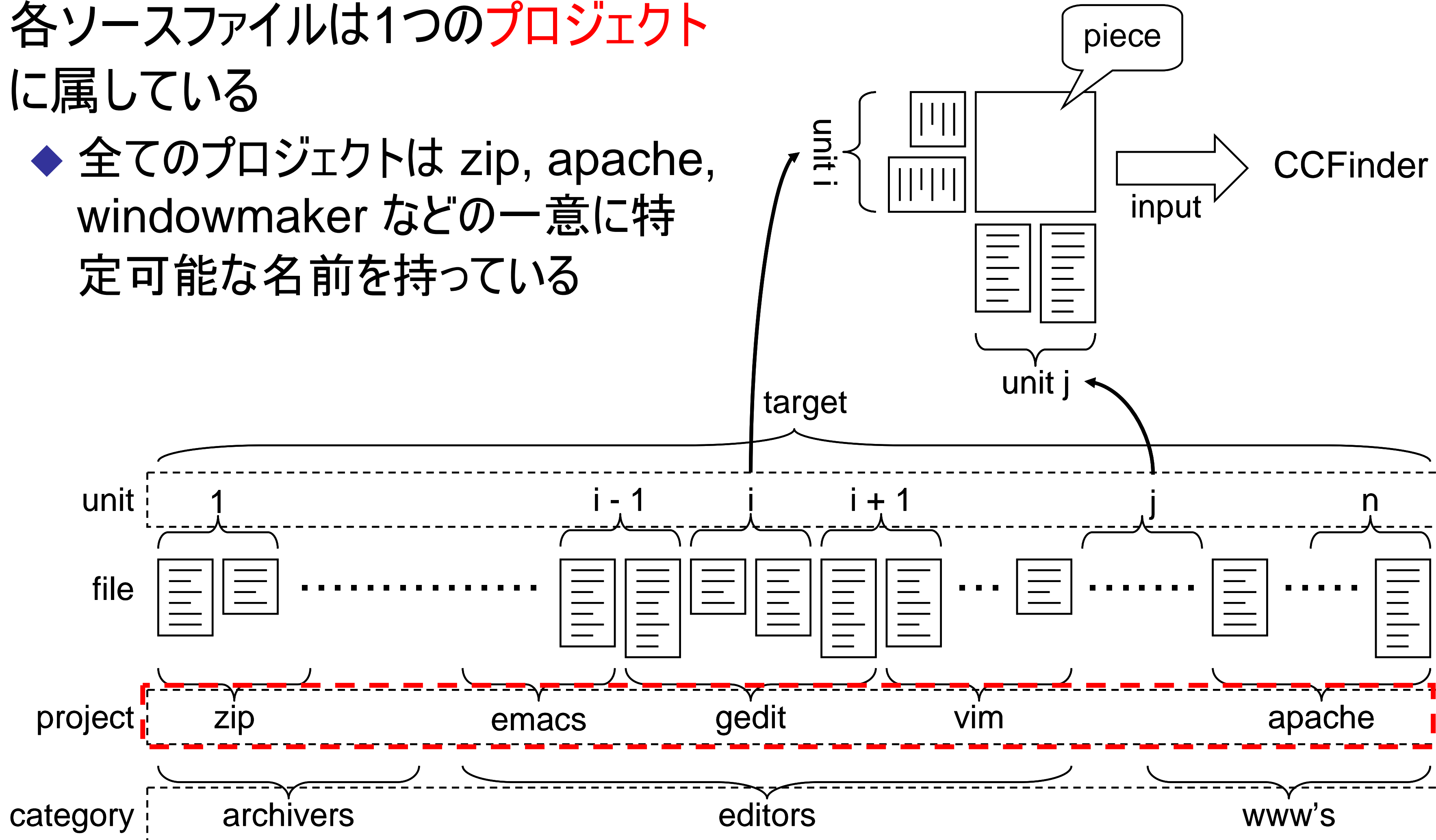
アプローチ

Software Engineering Laboratory

- 検出対象は非常に大規模なオープンソースソフトウェアの集合
  - 例：FreeBSD用のソフトウェア集合 Ports に含まれるC言語で記述されたソフトウェア
    - 容量：10.8Gbytes
    - 総行数：約4億行
    - .c ファイル数：約750万
    - ソフトウェア数:約6,700
  - 以降，オープンソースターゲットと呼ぶ
- 単一のコンピュータ上で検出を行うことは現実的ではない
  - 検出対象を細かく分割してCCFinderを実行した場合，約40日を要するとの予測
- 分散処理方式を用いることにより，高速にコードクローン検出を行う

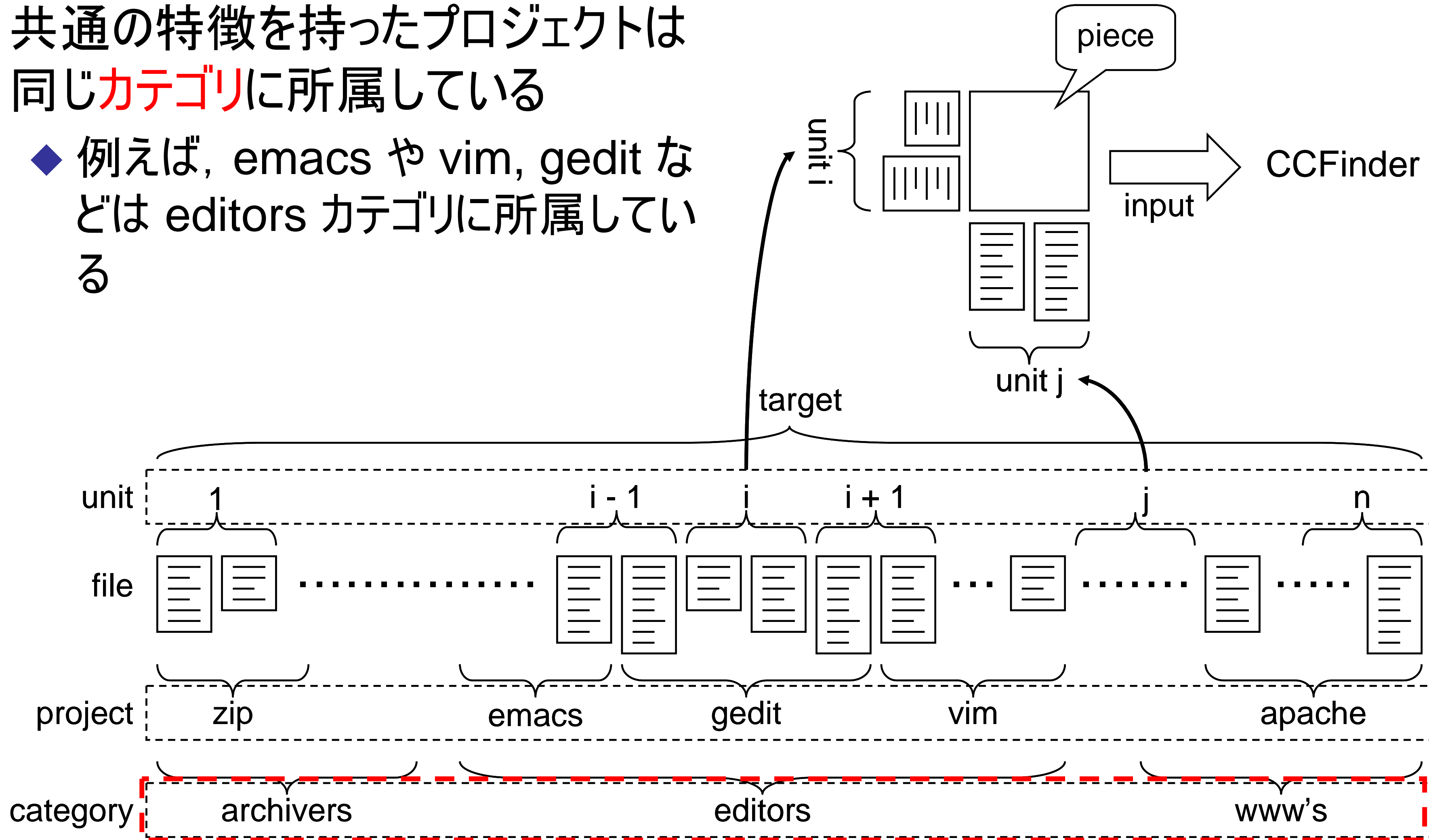
検出対象

- 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, apache, windowmaker などの一意に特定可能な名前を持っている



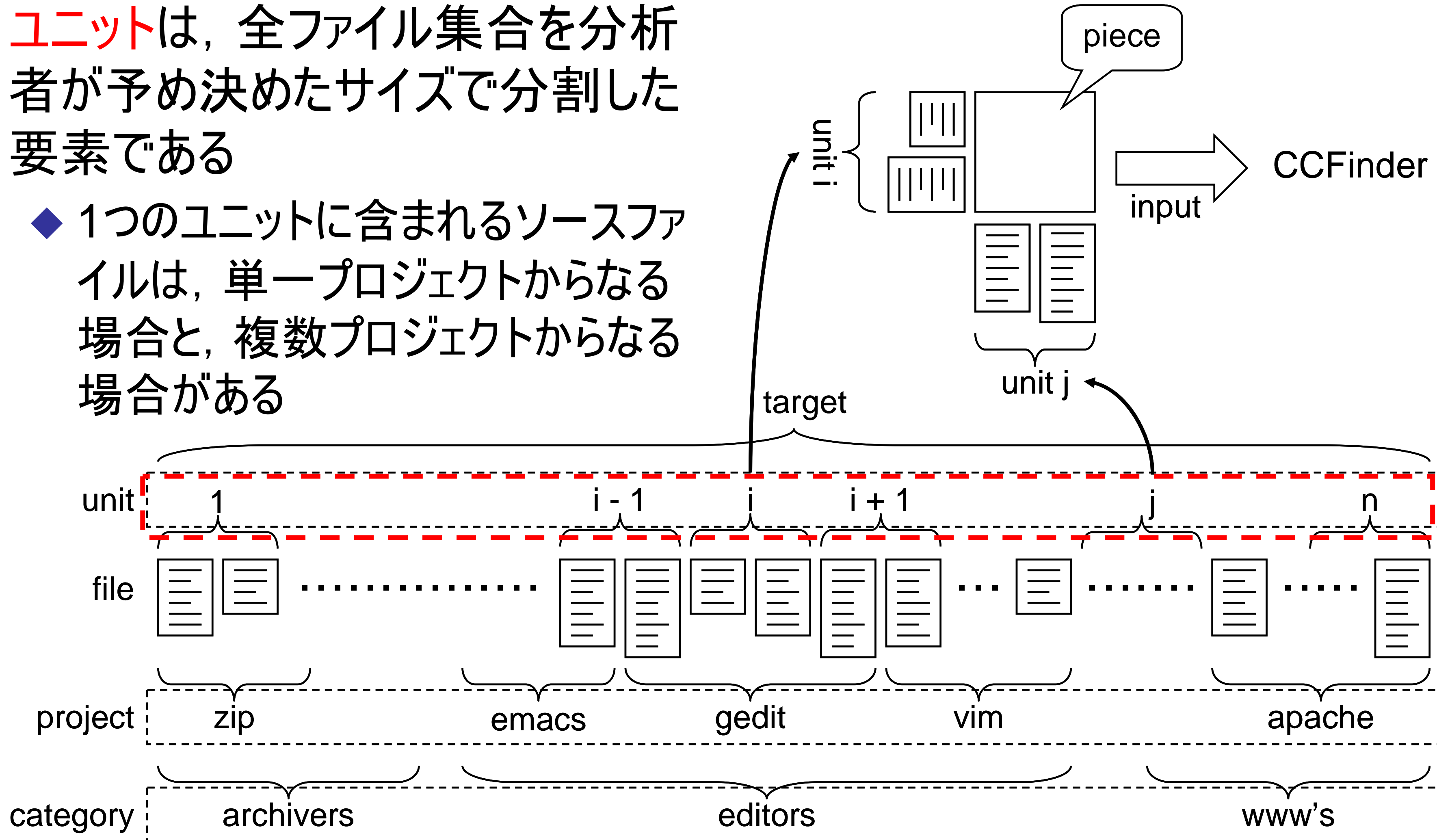
検出対象

- 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
  - 例えば，emacs や vim, gedit などは editors カテゴリに所属している



検出対象

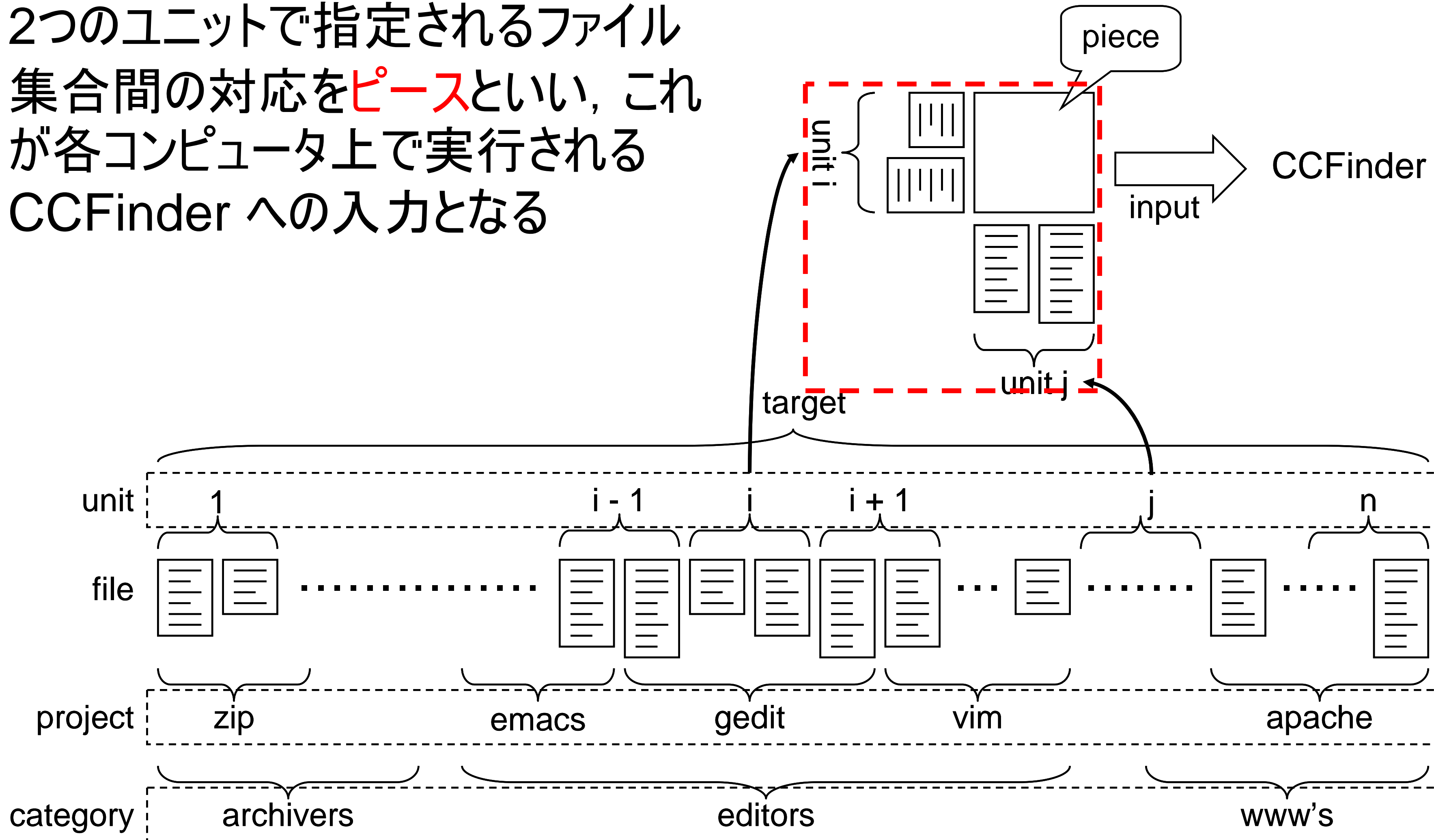
- ユニットは，全ファイル集合を分析者が予め決めたサイズで分割した要素である
  - 1つのユニットに含まれるソースファイルは，単一プロジェクトからなる場合と，複数プロジェクトからなる場合がある



検出対象

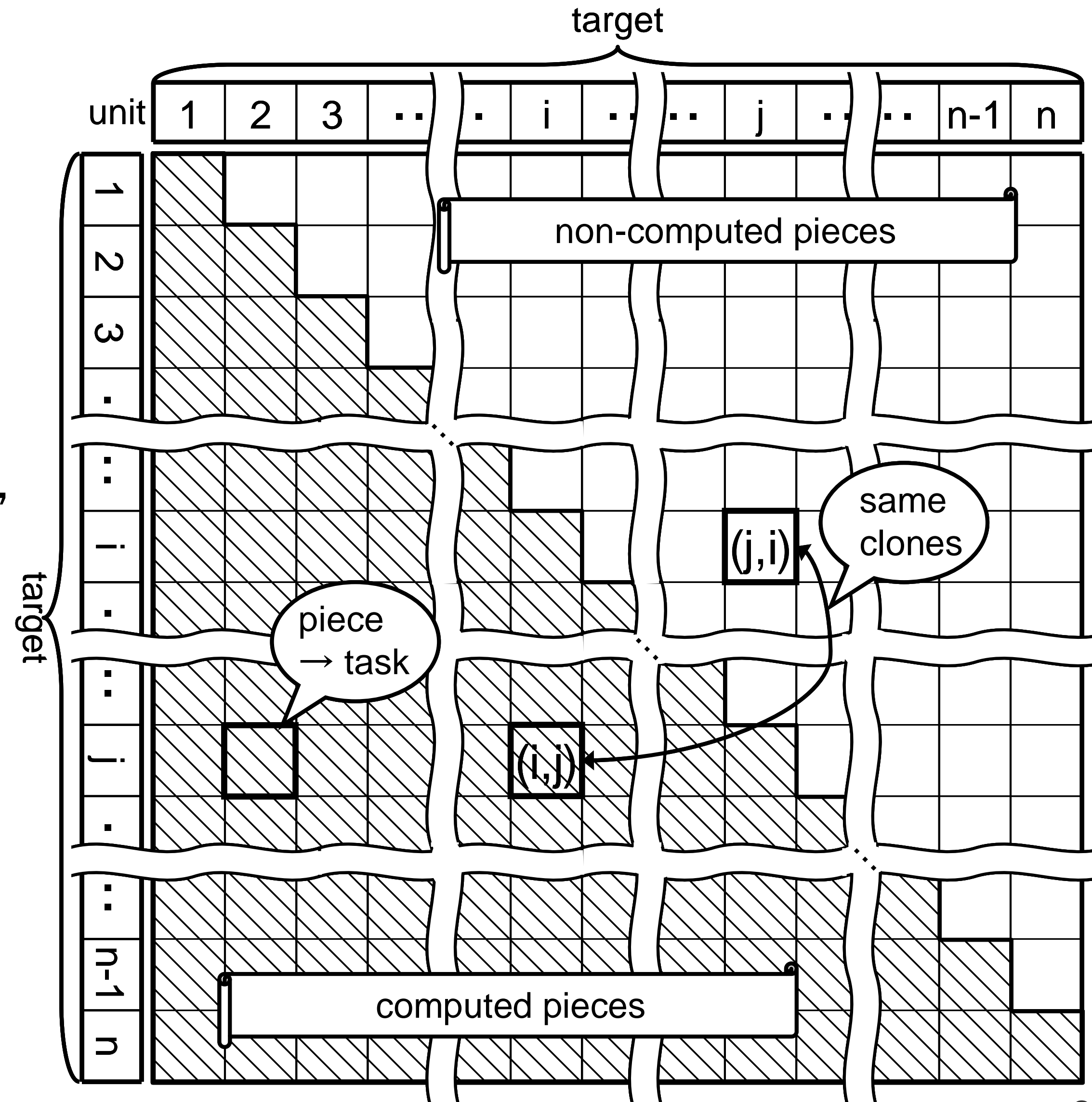
Software Engineering Laboratory

- 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

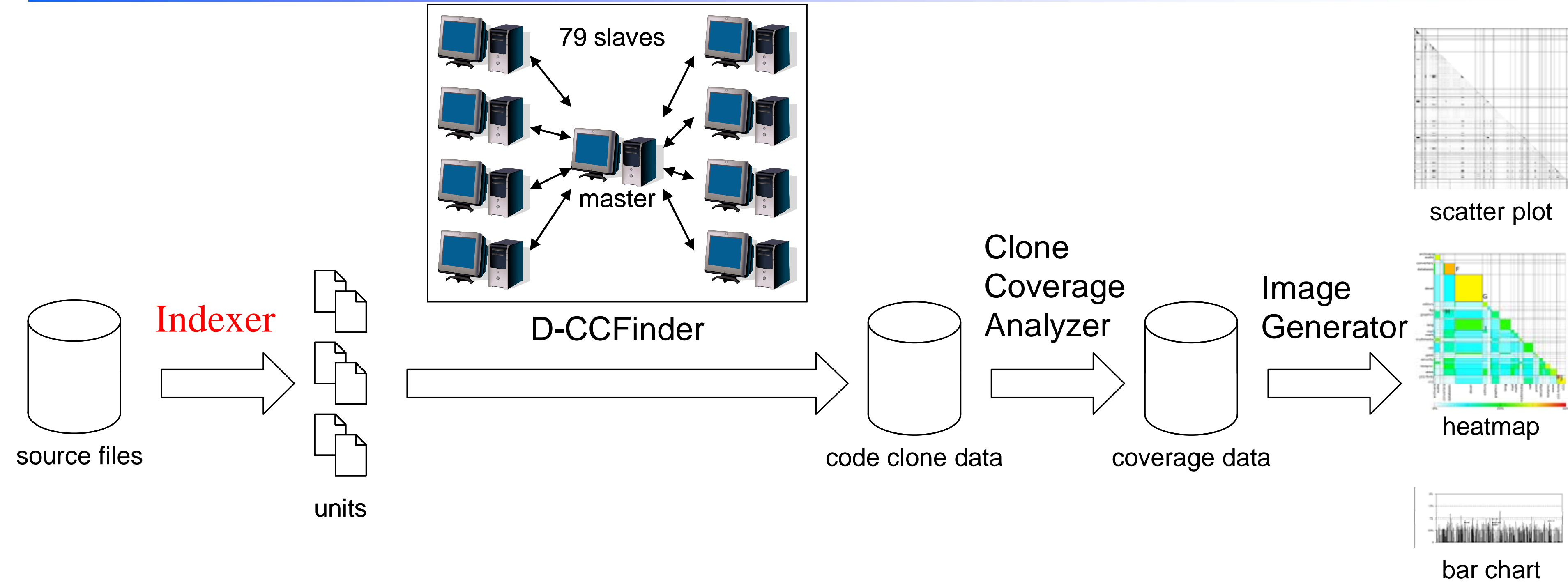


計算モデル

- 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
- ピース(i, j)と(j, i)に含まれるコードクローンは等価であるため，後者については，検出しない
- 各ピースの演算は他のピースの演算に全く依存しない
  - 分散処理に適している

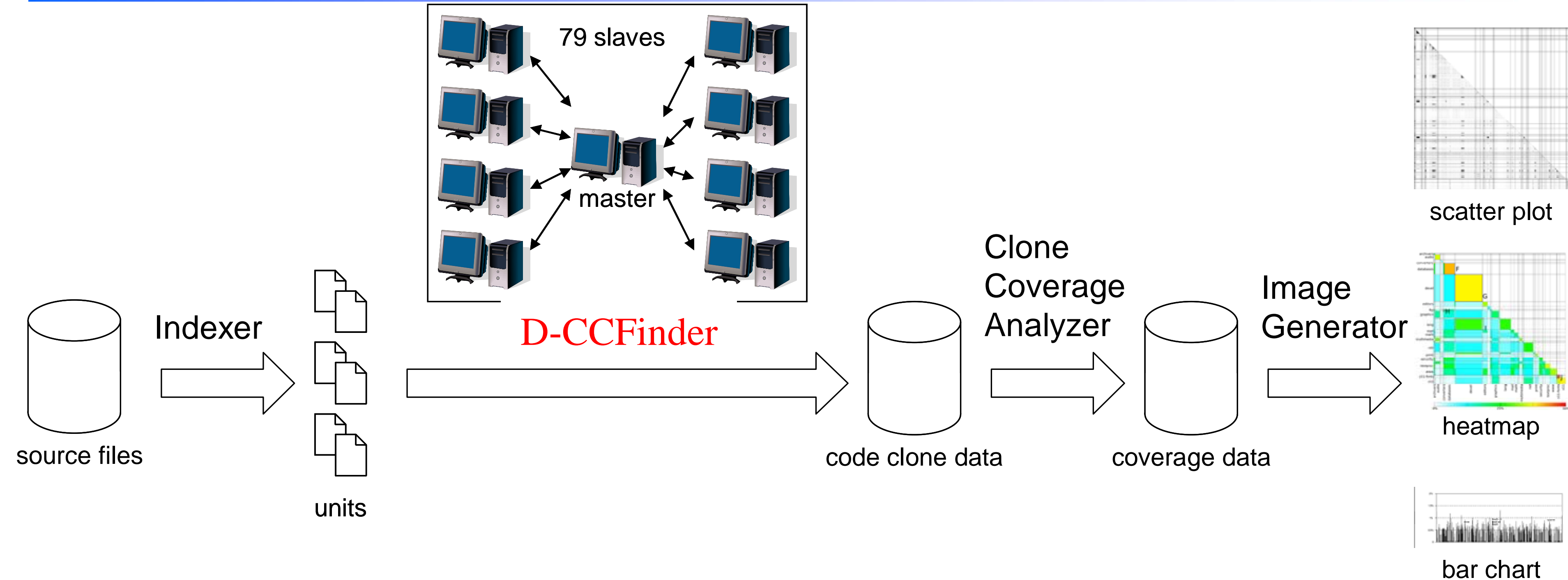


D-CCFinder システム Indexer



- 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

D-CCFinder システム
D-CCFinder(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新たなピースを作成し，割り当てる

D-CCFinder システム
D-CCFinder（スレーブノード）

Software Engineering Laboratory

79 slaves

master

Indexer

D-CCFinder

Clone Coverage Analyzer

Image Generator

source files

units

code clone data

coverage data

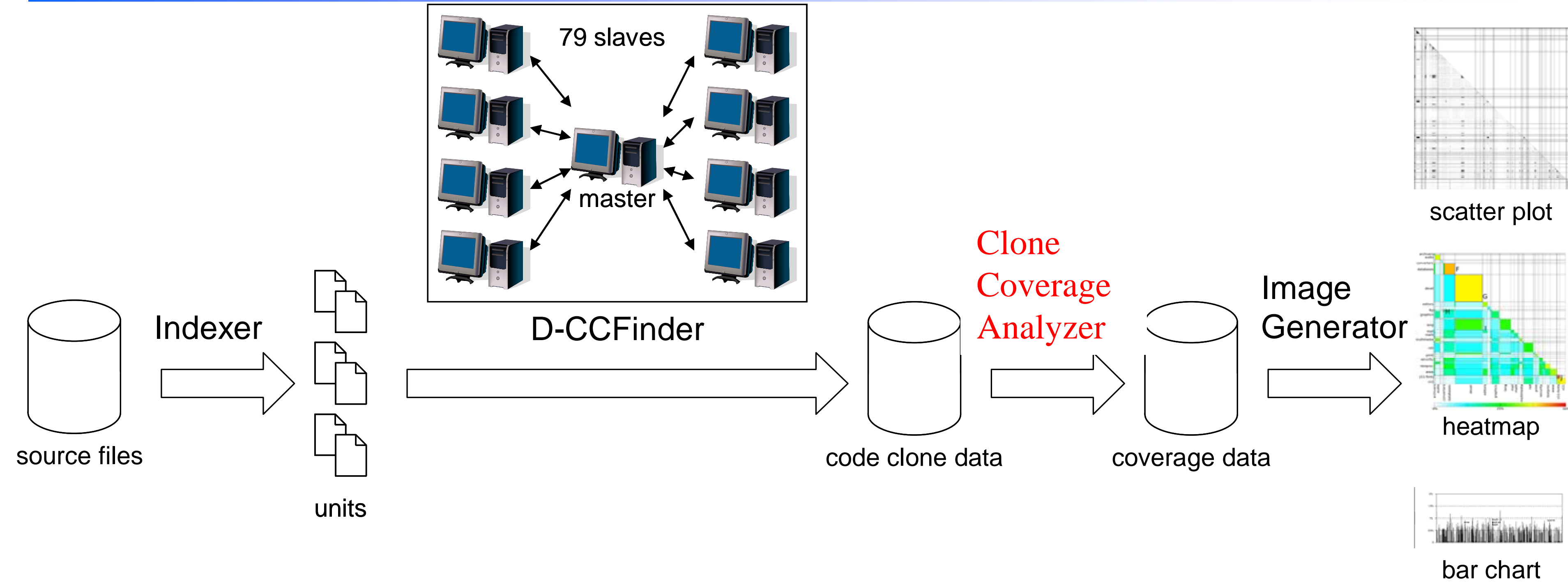
scatter plot

heatmap

bar chart

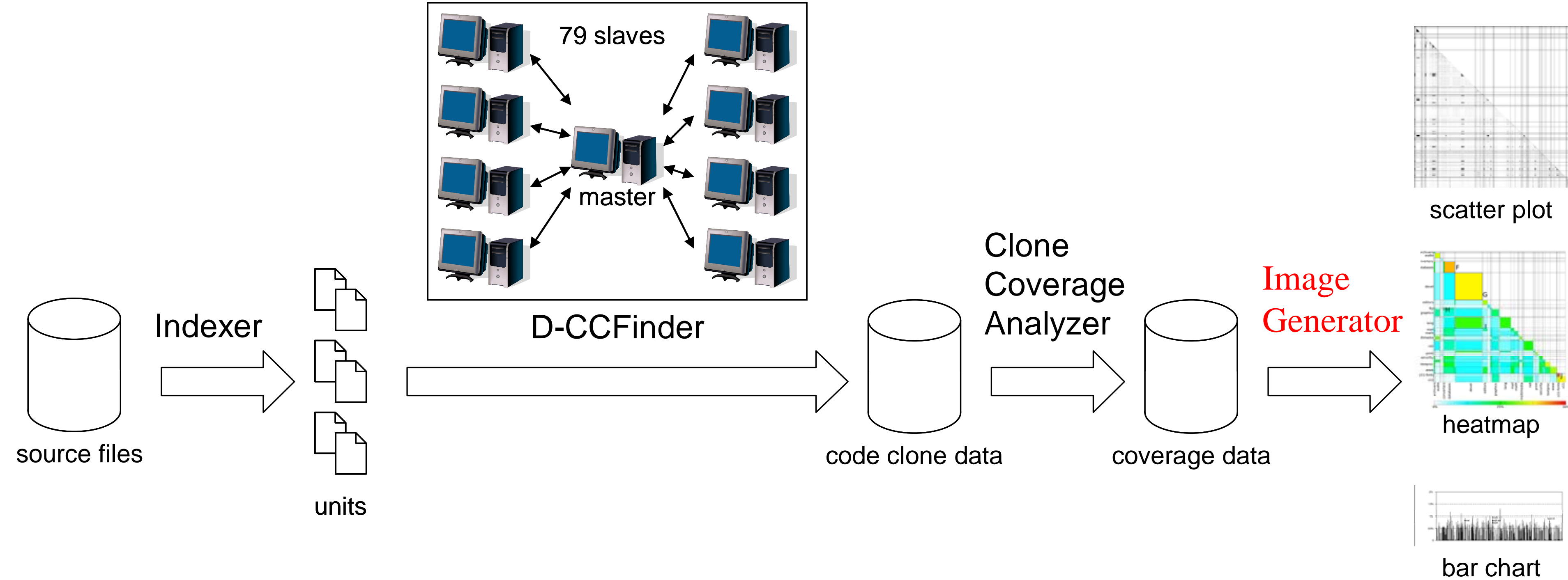
- マスターノードから割り当てられた入力ファイル（ピース）を用いてコードクローン検出処理を行う.
- 検出対象ファイルはスレーブノードのローカルファイルシステムにコピーされる.
- 検出処理後もローカルファイルシステム上のコピーは削除されず，次回以降の検出処理のキャッシュとして利用される

D-CCFinder システム
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

D-CCFinder システム Image Generator



- Clone Coverage Analyzer が生成した定量的な情報を用いて，散布図やヒートマップ等を生成する

D-CCFinderシステム 計算機・ネットワーク環境

Software Engineering Laboratory

- D-CCFinderを学生演習室のマシン80台上で構築した
  - 1台がマスター，残りの79台がスレーブ
- マシンスペック
  - CPU: PentiumⅣ 3.0GHz
  - Memory: 1.0GBytes
  - Local storage: 40-50GBytes
- 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
  - 通信はJava RMIを用いて行う
- 検出対象ソースファイル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する
  - 各マシンはNFS経由でアクセスする

実験

- オープンソースターゲットを用いて2種類の実験を行った
  - 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

- 実験１：オープンソースターゲット内のコードクローンの状態を調査
  - プロジェクト間のコードクローンのみを検出（プロジェクト内に閉じたコードクローンは検出していない）
- 実験２：井上研究室で過去に開発したソフトウェア（SPARS-J）とオープンソースターゲット間の類似度調査
  - SPARS-J: 約47,000行

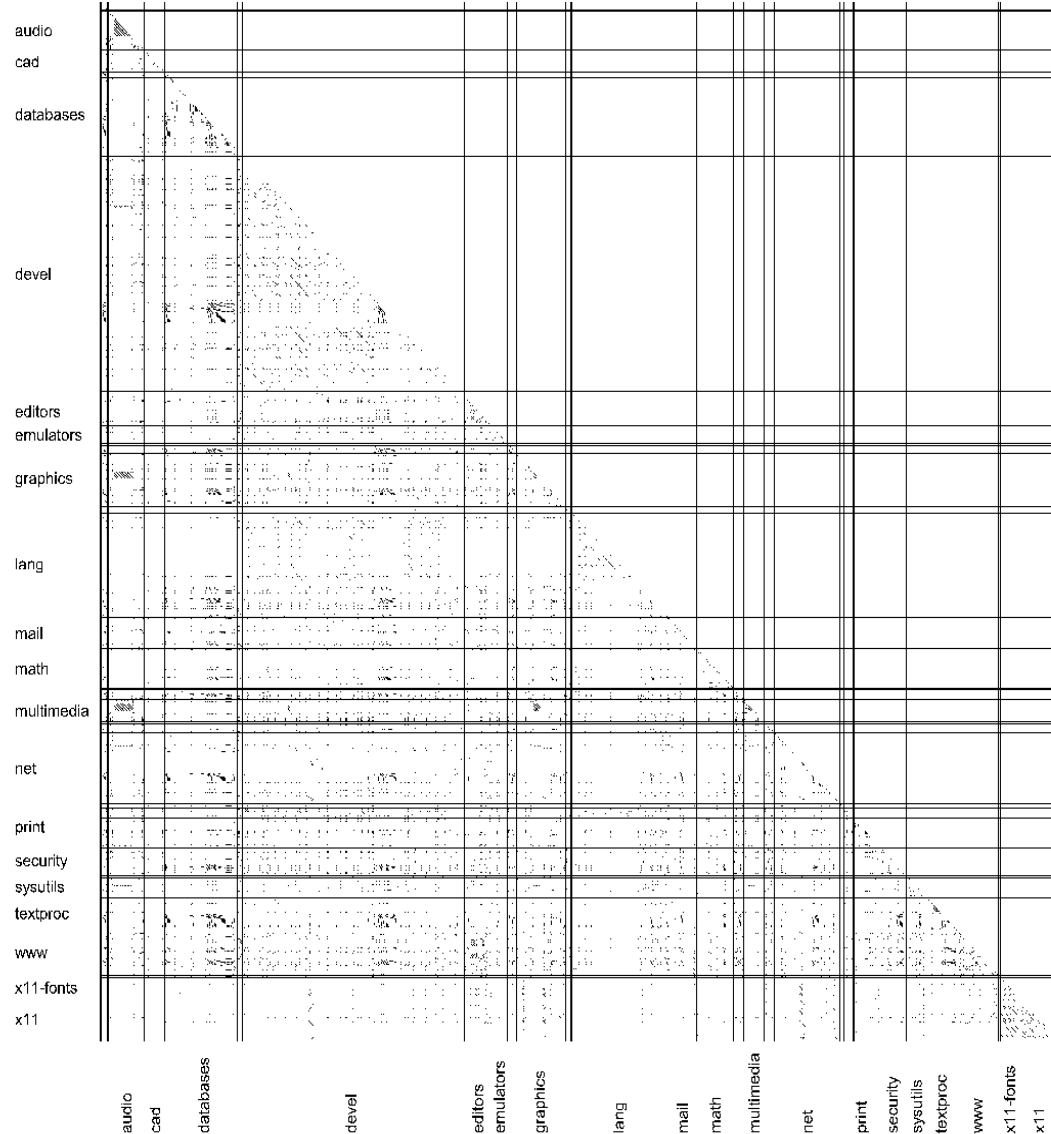
実験1

- 最小一致トークン数（CCFinderが検出するクローンの閾値）を50, ユニットサイズを15MBytesに設定して，D-CCFinderを実行した.
  - 実行されたタスクの総数は 269,745個
- 80台のコンピュータ上でD-CCFinderを実行した結果，約51時間でコードクローン検出を完了することができた
  - この検出速度は，単一のコンピュータ上で行った場合の約20倍
  -

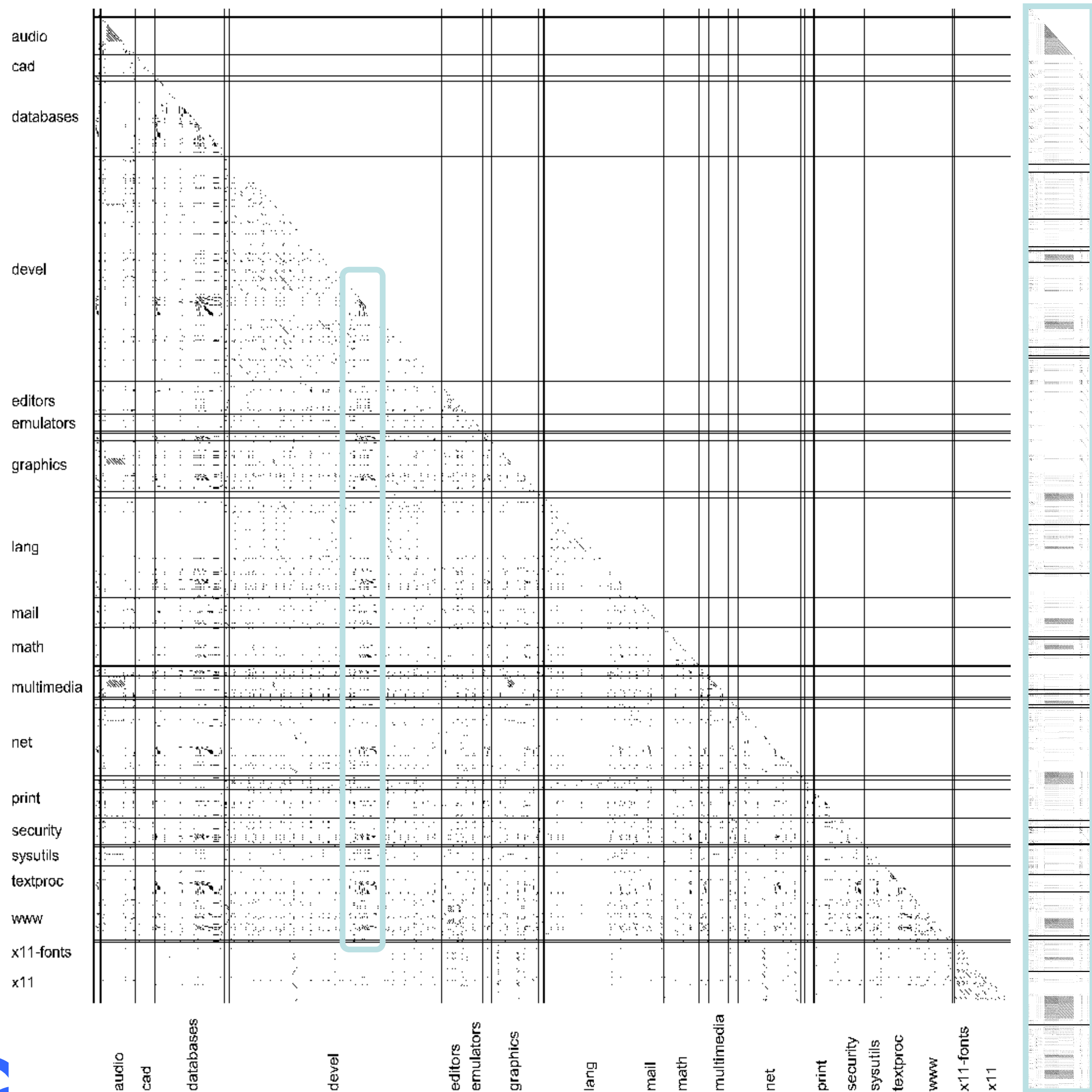
実験1：散布図を用いた調査(1/4)

Software Engineering Laboratory

- 右図はオープンソースターゲット全体の散布図
  - 1ピクセルが200x200のファイルを表している
  - 1つでもクローンが含まれていた場合は黒，全く含まれていない場合は白で描画



実験1：散布図を用いた調査(2/4)



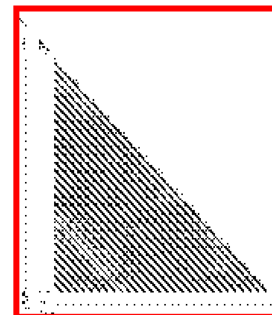
php4/5 のソースコードの流用．様々なカテゴリに含まれるソフトウェアに流用されている

実験1：
散布図を用いた調査(3/4)

Software Engineering Laboratory

audio
cad
databases
devel
editors
emulators
graphics
lang
mail
math
multimedia
net
print
security
sysutils
textproc
www
x11-fonts
x11

audio cad databases devel editors emulators graphics lang mail math multimedia net print security sysutils textproc www x11-fonts x11

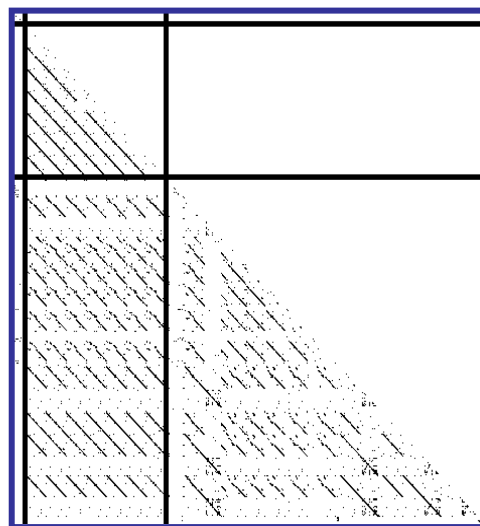


カテゴリaudio内に存在するマルチメディアフレームワークgstreamerとその複数のプラグインが多くの同一ファイルを所持していた

実験1：
散布図を用いた調査(4/4)

audio
cad
databases
devel
editors
emulators
graphics
lang
mail
math
multimedia
net
print
security
sysutils
textproc
www
x11-fonts
x11

X11関係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のコピーであった



実験1：
ヒートマップを用いた調査(1/5)

- より正確に重複度の大きい部分を調査するためにヒートマップを用いた調査をおこなった

- 右図はカテゴリレベルのヒートマップ

- カバレッジの定義

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

$LOC(M)$: モジュールMの行数

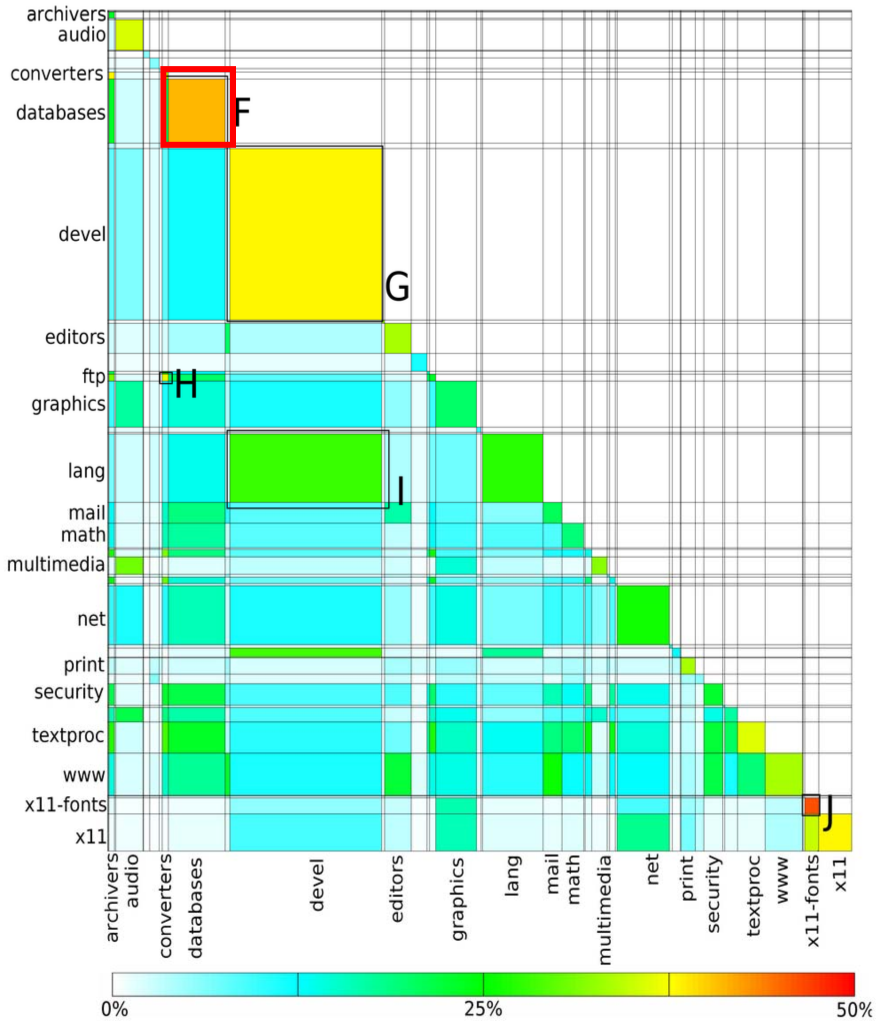
$C_{M_0}(M_1)$: モジュールM_0のうち，
モジュールM_1とクローンになっている部分

archivers
audio
converters
databases
devel
editors
ftp
graphics
lang
mail
math
multimedia
net
print
security
textproc
www
x11-fonts
x11

F
G
H
I
J

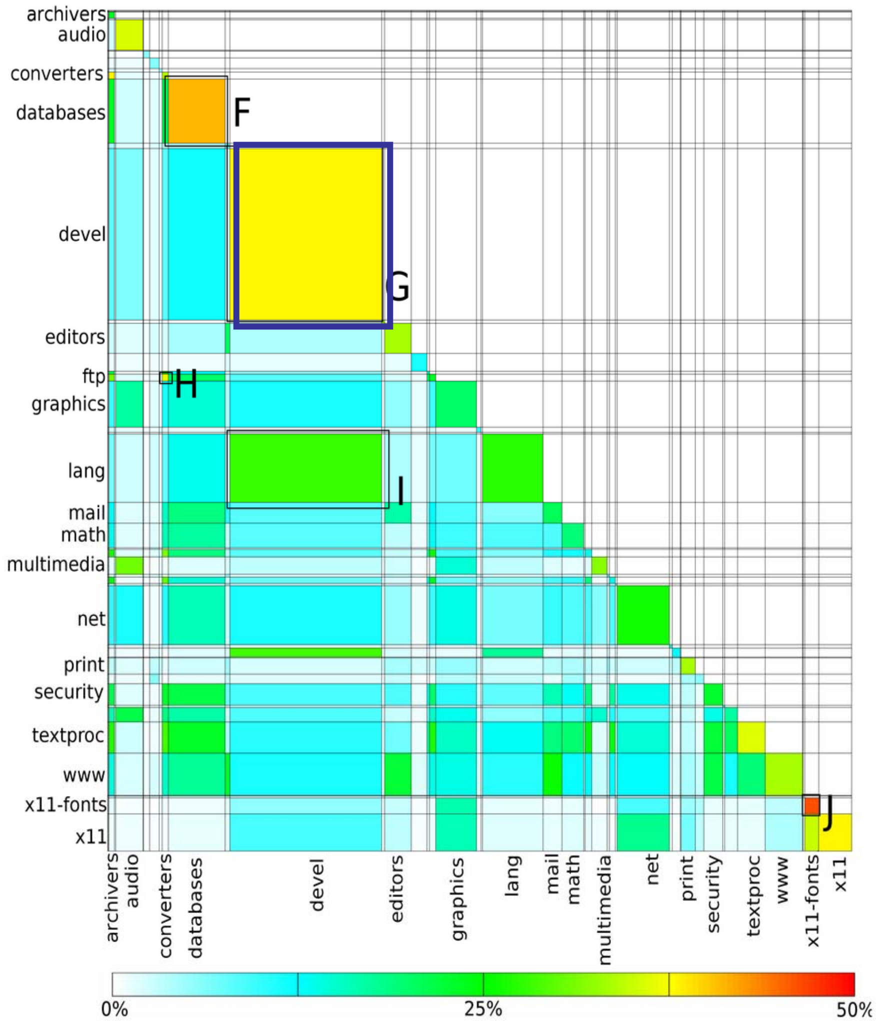
0%
25%
50%

実験1：
ヒートマップを用いた調査(2/5)



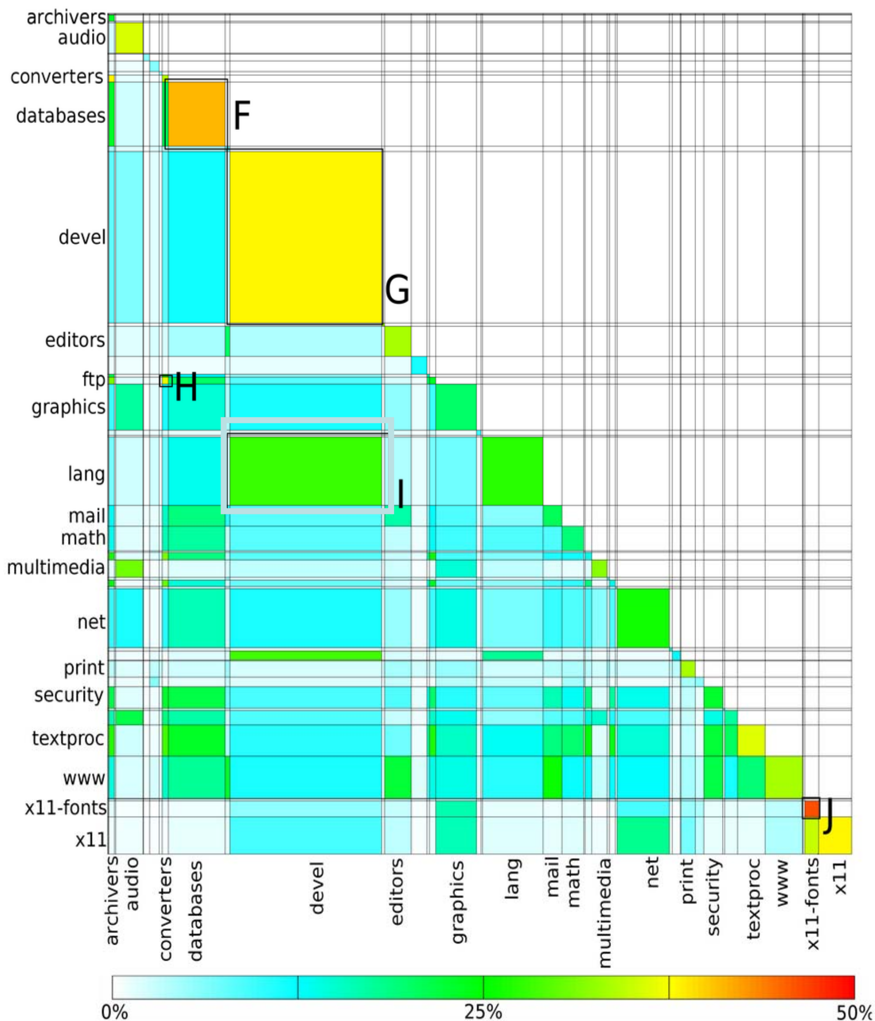
databases カテゴリ内
カバレッジ: 41%
クローンの原因:
- PostgreSQLなどいくつかのソフトウェアの複数バージョンが存在
- さまざまなプログラミング言語からデータベースを操作するためのドライバの存在
- php4/5のソースの流用

実験1：
ヒートマップを用いた調査(3/5)



devel カテゴリ内
カバレッジ: 38%
クローンの原因:
•GNU binary utities (binutiils) とコンパイラの多くのバージョンが存在したため

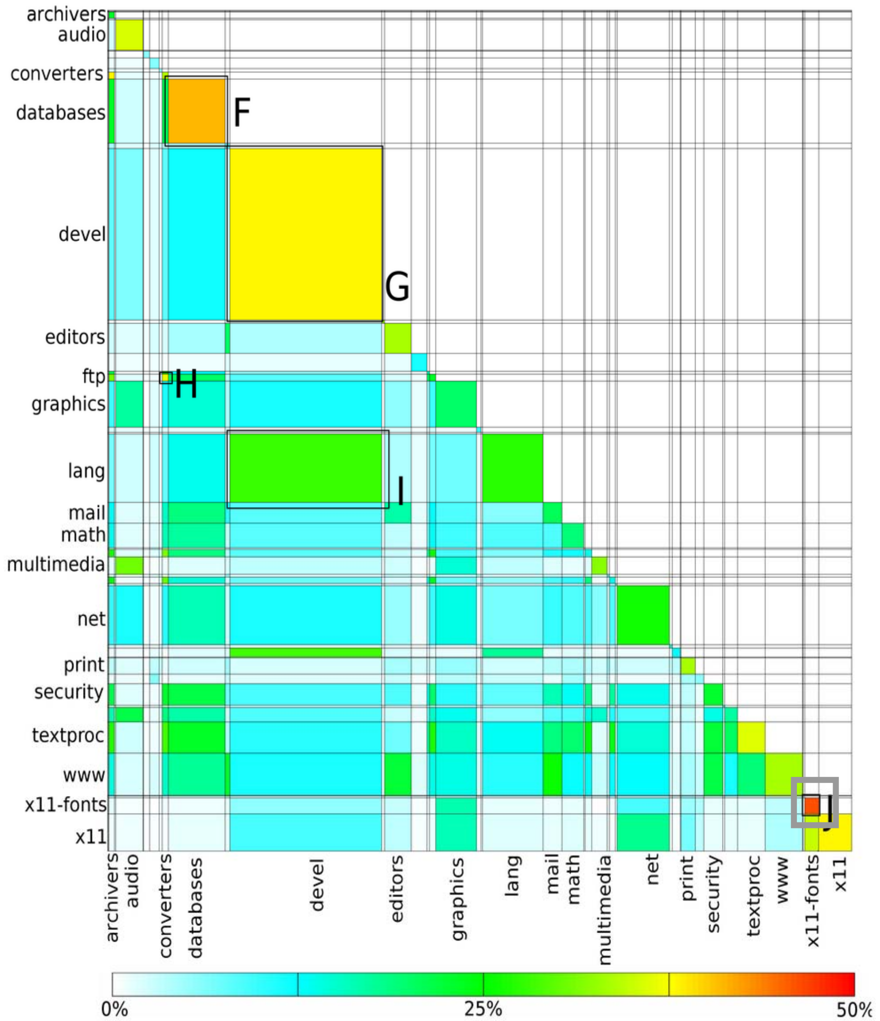
実験1：
ヒートマップを用いた調査(4/5)



lang と devel カテゴリ間
カバレッジ: 28%
クローンの原因:
•devel内にGNU コンパイラの複数のバージョンが存在しており，lang に含まれるソフトウェアがそのコードを流用していたため

実験1：ヒートマップを用いた調査(5/5)

Software Engineering Laboratory



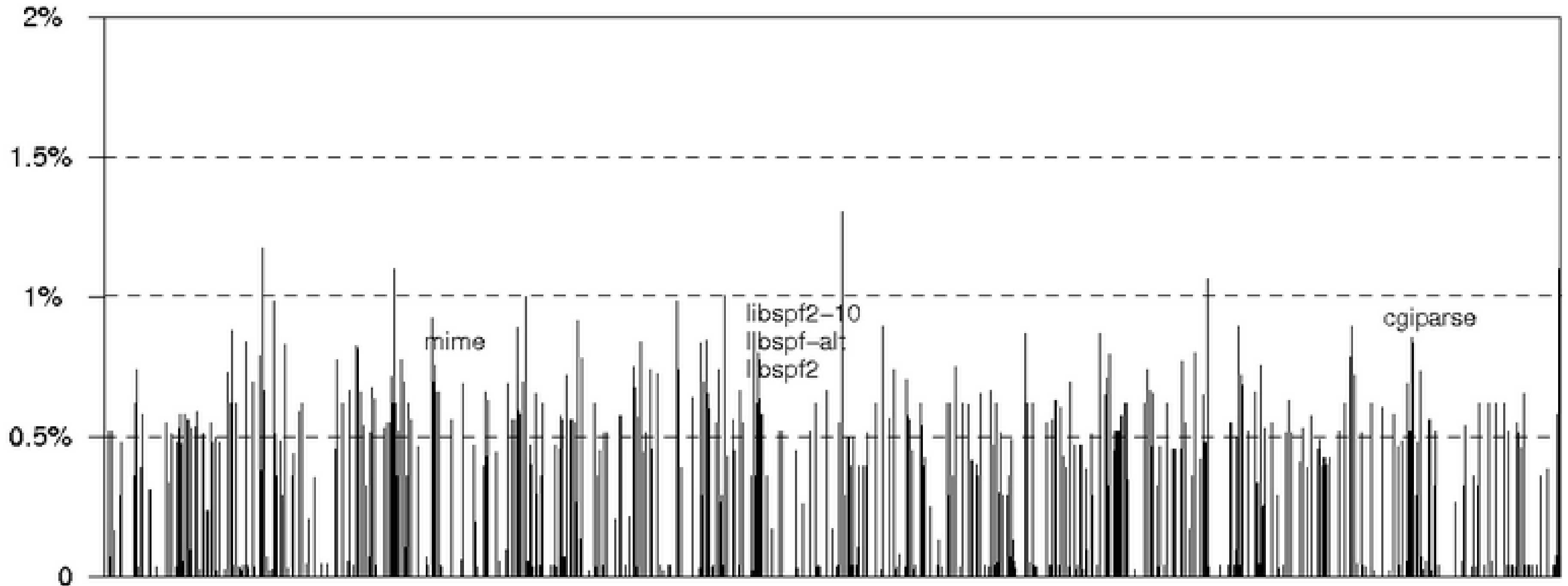
x11-fonts カテゴリ内
カバレッジ: 46%
クローンの原因:
- X11関係のソフトウェアと多くのコードが重複していた
- カテゴリサイズが小さいこともカバレッジが高い要因となっている

実験2

- 最小一致トークン数（CCFinderが検出するクローンの閾値）を50, ユニットサイズを15MBytesに設定して，D-CCFinderを実行した.
  - 実行されたタスクの総数は 734個
- 80台のコンピュータ上でD-CCFinderを実行した結果，約40分でコードクローン検出を完了することができた

実験2
棒グラフによる調査（フィルタリング前）

Software Engineering Laboratory



クローンの多くが getopt.c という名前のファイルに含まれていた

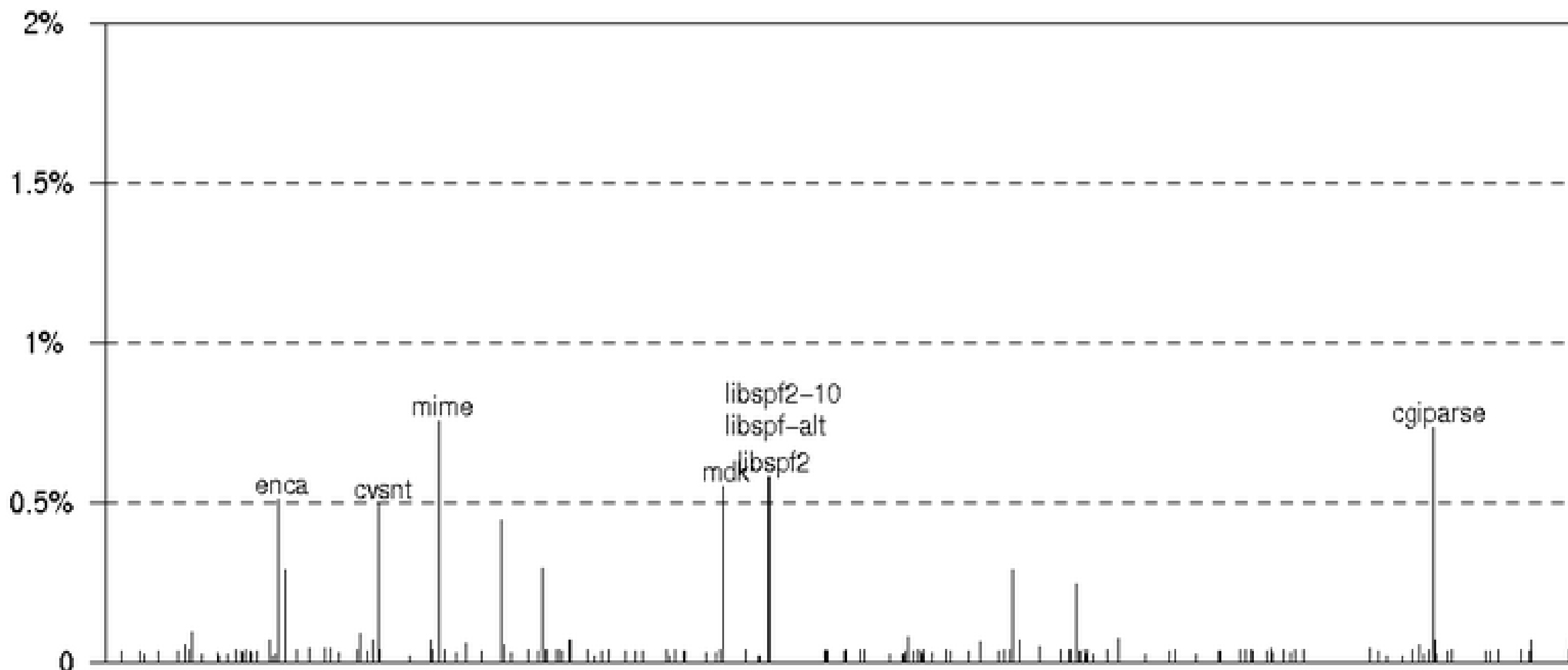
$$Coverage(M) = \frac{LOC(C_M(SPARS-J))}{LOC(M)}$$

$LOC(M)$：モジュールMの行数

$C_M(SPARS-J)$：モジュールMのうち，$SPARS-J$とクローンになっている部分

実験2
棒グラフによる調査（フィルタリング後）

getopt.c という名前のファイルを除外した



クローンになっている部分

- CGI処理の部分
- getopt.c に独自のカスタマイズが行われているもの

考察

Software Engineering Laboratory

- 80台のコンピュータ上でD-CCFinderを実行した結果，約51時間でコードクローン検出を完了することができた
  - 理論上は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出力の後処理などのオーバヘッドが発生した
    - ギガビットスイッチなどの導入によりさらに高速化可能
- 散布図をより正確に生成する必要がある
  - 現在は速度とサイズを重視しているために精度が悪い（1ピクセルは200x200）ため，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ができない
- SPARS-Jはオープンソースとあまりコードクローンを共有していないことがわかった．また，コードクローンを共有しているファイルを突き止め，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
  - ソフトウェアの著作権違反調査に応用することができると考えられる

まとめ

- 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
- 提案した手法を D-CCFinder システムとして実装し，オープンソースターゲットを対象として適用実験を行った
  - D-CCFinderは既存のネットワーク環境上で実装されている
- 散布図，ヒートマップ，棒グラフなどを用いてコードクローンの状態を把握することができた