Translating imperative code to MapReduce

Cosmin Radoi — University of Illinois

Stephen J. Fink, **Rodric Rabbah** — IBM T.J. Watson Research Center

Manu Sridharan — Samsung Research America

motivation

Why translate to MapReduce?
- parallel, distributable programming model
- fault-tolerance, elastic scaling
- integration with distributed file system
- popular ecosystem — many good tools and services

Why translate *automatically* to MapReduce?
- although simple, MapReduce is not easy
- reduce cost of retargeting legacy code
- allow developers to concentrate on familiar sequential code

challenges

Parallelization
- loop-carried dependencies
- mappers and reducers can only access *local* data (MapReduce)

Imperative input code
- MapReduce is conceptually functional

Indirect memory access
- mappers and reducers communicate via a *shuffle* operation
- the shuffle is usually equivalent to an indirect memory access
- indirect memory accesses are hard for parallelizing compilers

sequential imperative

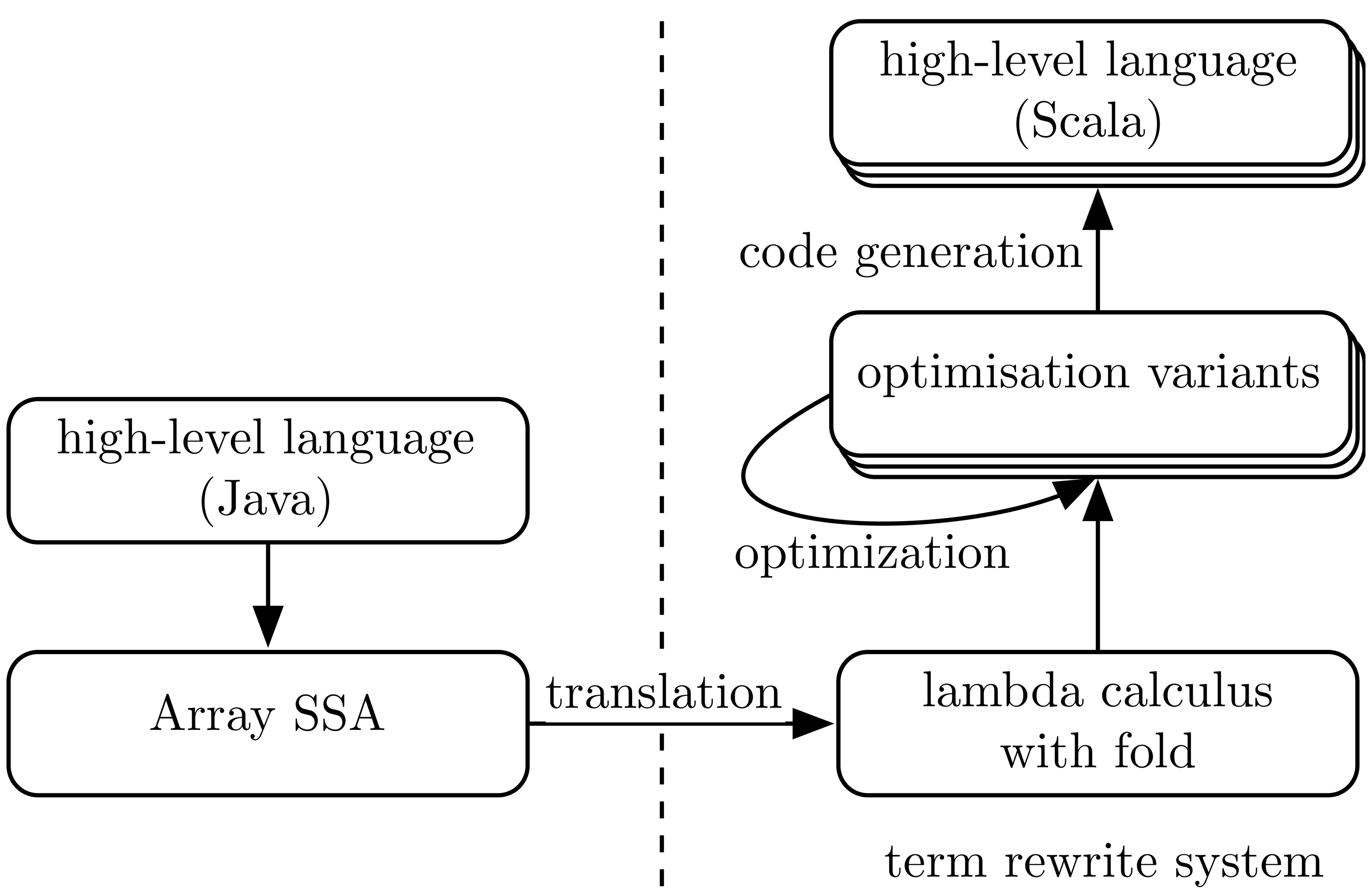
```
Map<String,Integer> count = new HashMap<>();

for (int i = 0; i < docs.size(); i++) {
  String[] words = tokenize(docs.get(i));
  for (int j = 0; j < words.length; j++) {
  String word = words[j];
    Integer prev = count.get(word);
    if (prev == null) prev = 0;
    count.put(word, prev + 1);
  }
}
```

⇒ **MOLD**

functional MapReduce

```
docs
  .flatMap({ case (i, doc) => tokenize(doc) })
  .map({ case (j, word) => (word, 1) })
  .reduceByKey({ case (c1, c2) => c1 + c2 })
```

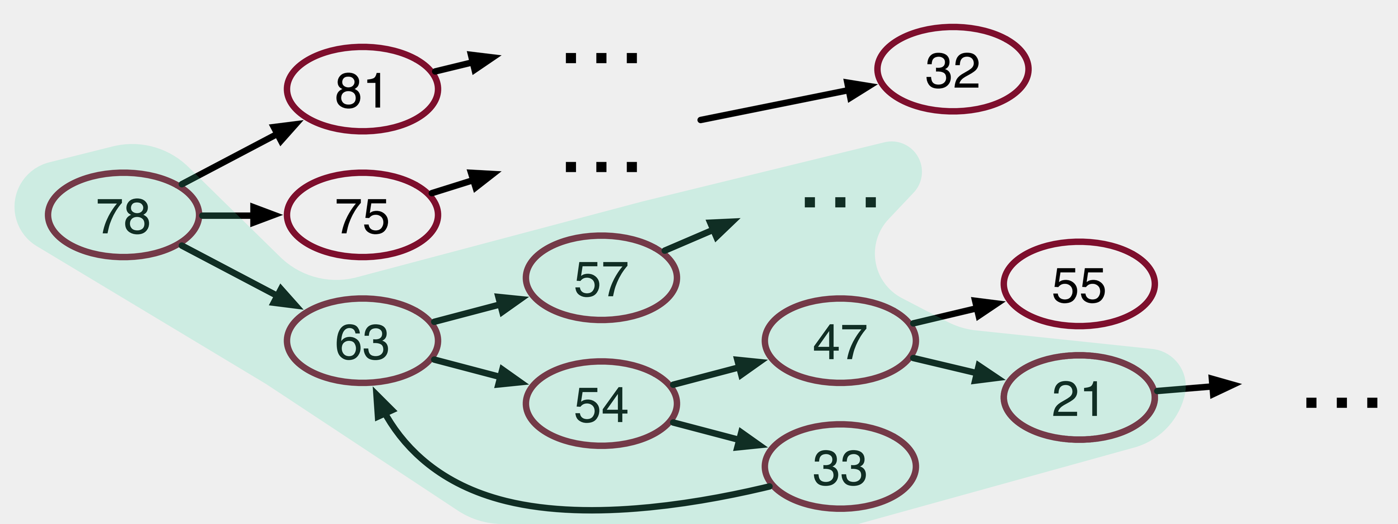
system overview



program variant exploration

Equivalent states
- all α-equivalent which have the same β-reduced form
- rewriting over β-reduced form — no recursion

Search
- based on pluggable cost function
- cost function can be platform-dependent
- gradient descent



fold-to-groupBy

```
words.fold(count){ case (runningCount, word) =>
    runningCount.update(word, runningCount(word) + 1)
}
```

⇒

```
words.groupBy(word => word).map { case (word, list) =>
        list.fold(count(word)) { (sum, elem) => sum + 1 }
}
```

Pattern matching variables: `domain` `collection` `index expression` `value expression`

Condition: the value expression only accesses the collection at the index expression

evaluation

Evaluation suite
- Phoenix benchmark suite
- WordCount, Color Histogram, LinearRegression, StringMatch, MatrixProduct, Principal Component Analysis (PCA), K-Means

results

Can MOLD generate *effective* MapReduce code?
- no redundant computation — 5/7 programs
- parallelism — optimal for 4/7 programs
- memory accesses are localized — 5/7 programs

OOPSLA '14 — talk is Friday @ 2:37pm, Salon F