**Map-Reduce in Various Programming**

**Languages**

**CSInParallel Project**

July 26, 2012

**CONTENTS**

**1 Context of Map-Reduce Computing 1**

**2 Example: WebMapReduce using Scheme language 2**

**3 WebMapReduce in various languages 3**

3.1 Scheme . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

3.2 C++ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

3.3 Java . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

3.4 Python . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

3.5 Comparison . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

**4 Accessing concurrency and parallelism within a programming language 8**

4.1 Features that are part of a programming language . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

4.2 Libraries . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

4.3 Other approaches . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

**CHAPTER**

**ONE**

**CONTEXT OF MAP-REDUCE**

**COMPUTING**

• The use of LISP’s map and reduce functions to solve computational problems probably dates from the 1960s –

very early in the history of programming languages

• In 2004, Google published their adaptation of the map-reduce strategy for data-intensive scalable computing (DISC) on large clusters. Their implementation, called **MapReduce**, incorporates features automatically to split up enormous (e.g., multiple *petabytes*) data sets, schedule the mapper and reducer processes, arrange for those processes always to operate on local data for performance efficiency, and recover from faults such as computers or racks crashing.

• MapReduce, together with the page rank algorithm, gave Google the competitive combination it needed to become the most popular search engine (approximately 2/3 of the market at present). Google proceeded to apply map-reduce techniques to everything from ad placement to maps and document services.

• Google’s MapReduce is proprietary software. But Yahoo! created the Hadoop implementation of this map- reduce strategy for clusters as an Apache Software Foundation open-source project. Consequently, Hadoop is used not only at Yahoo!, but at numerous other web service companies, and is available for use at colleges and universities.

• *Future systems:* (1) Strategies such as map-reduce that enable programmers to provide relatively simple code segments and reuse code for synchronization, fault tolerance, etc., are a target for forthcoming systems (*View from Berkeley*, 2006). (2) Future systems are likely to consist of multiple heterogeneous cores, programmed using functional programming techniques (Michael Wrinn, Intel, keynote speech at SIGCSE 2010).

**EXAMPLE: WEBMAPREDUCE USING**

**SCHEME LANGUAGE**

• **WebMapReduce (WMR)** is a strategically simplified interface for performing map-reduce computing devel- oped by students from St. Olaf College. While initially supporting Scheme, the platform currently supports several high-level languages, including Python, C++, and Java.

• Wmr\_scm.pdf includes specs for the functions provided in this Scheme WMR interface. download

Wmr\_scm.pdf

• This implementation of the Scheme interface for entering mappers and reducers uses an iterator for providing values in a reducer. Each call of a reducer receives all the key-value pairs for a particular key, and the two arguments for that reducer are that key and an iterator for obtaining the values.

**Iterator** - an (object-oriented programming) object that enables a programmer to obtain each value in a collection as a sequence of values, encapsulating the internal representation of that collection. We may visualize an iterator as a “dispenser” of values, providing one value at a time until all are exhausted.

• Iterators are used to provide reducer values because when there are very many key-value pairs, the total size of the collection of values may exceed the size of main memory.

• As the spec indicates, the second argument of a reducer is a WmrIterator object (we’ll call it iter), and that object iter has two methods:

**–** The call (iter ’has-next) returns true if a next element exists, false otherwise.

**–** The call (iter ’get-next) delivers the next element from the iterator, and advances that iterator (so a next call to the iterator will return a fresh value, if available); this call returns false if there is no next value.

**WEBMAPREDUCE IN VARIOUS**

**LANGUAGES**

The following subsections include the word count mapper and reducer implementations for WMR in several languages:

**3.1 Scheme**

Word count for WMR in Scheme language (spec is found on Wmr\_scm.pdf)

**3.1.1 mapper**

(**define** helper

(**lambda** (lis) *; a list of strings*

(**cond**

((null? lis) #t)

*;; at least one element in lis*

(**else** (wmr-emit (car lis) "1") (helper (cdr lis))))))

(**define** mapper

(**lambda** (key value)

(helper (wmr-split key))))

**3.1.2 reducer**

(**define** loop

(**lambda** (key iter ct) *; a key,value iterator in its current state, and*

*; subtotal so far*

(**cond**

((iter ’has-next)

(loop key iter (+ ct (string->number (iter ’get-next)))))

*;; assert -- no more input values to add*

(**else** (wmr-emit key (number->string ct))))))

(**define** reducer

(**lambda** (key iter) (loop key iter 0)))

**Note:** For this WMR interface for Scheme (see spec for details):

• As indicated before, the mapper and reducer in this Scheme interface are functions.

• String manipulation functions are primitive in Scheme, so a library function split is provided for this interface that allows one to specify delimiters by a regular-expression pattern. Type conversion is provided in Scheme through convenient (though long-named) functions number->string and string->number.

• We use Scheme-style objects as implemented at St. Olaf for the iterator for a reducer, as described above.

**3.2 C++**

Word count for WMR in C++ language (C++ style iterators, spec is found on Wmr\_cpp.pdf)

**3.2.1 mapper**

**class Mapper**

{

**public**:

**void** mapper(string key, string value)

{

**char** delim = ’ ’;

vector splits = Wmr::split(key, delim);

**for** (**unsigned int** i = 0; i < splits.size(); ++i)

{

Wmr::emit(splits.at(i), "1");

}

}

};

**3.2.2 reducer**

**class Reducer**

{

**public**:

**void** reducer(string key, WmrIterator iter)

{

**long** count = 0;

**while** (iter != WmrIterator::end())

{

count += Wmr::strToLong(\*iter++);

}

Wmr::emit(key, Wmr::longToStr(count));

}

};

**Note:**

for this WMR interface for C++ (see spec for details):

• The mapper and reducer are methods of classes Mapper and Reducer, respectively.

• Strings are split using the method Wmr::split() of a (predefined) library class Wmr. Rather than splitting on arbitrary regular expressions, the (required) second argument of Wmr::split() is a string of characters, any one of which counts as a delimiter. Type conversion between numbers and strings is not convenient in C++, so helper methods are provided.

• C++-style iterators are used in the reducer method. In this style of iterator, operator\* delivers the current value, operator++ is used to advance to the next value, and the end of an iterator is detected by comparing that iterator for equality with the special iterator value WmrIterator::end.

**3.3 Java**

Word count for WMR in Java language (Java style iterators, spec is found on Wmr\_java.pdf)

**3.3.1 mapper**

*/\* Mapper for word count \*/*

**class Mapper** {

**public void** mapper(String key, String value) { String words[] = key.split(" ");

**int** i = 0;

**for** (i = 0; i < words.length; i++) Wmr.emit(words[i], "1");

}

}

**3.3.2 reducer**

*/\* Reducer for word count \*/*

**class Reducer** {

**public void** reducer(String key, WmrIterator iter) {

**int** sum = 0;

**while** (iter.hasNext()) {

sum += Integer.parseInt(iter.next());

}

Wmr.emit(key, Integer.valueOf(sum).toString());

}

}

**Note:** for this WMR interface for Java (see spec for details):

• The mapper and reducer are again methods of classes Mapper and Reducer, respectively, as for C++.

• Java provides useful string manipulation methods. Type conversion is provided in the Java libraries, but is inconvenient.

• Java style iterators are used for the reducer. These have methods hasNext() which returns false when no new values exist in an iterator, and next() which returns the next unseen value and advances that iterator.

**3.4 Python**

Word count for WMR in Python3 language (Python3 style iterators, spec is found on Wmr\_jpy3.pdf)

**3.4.1 mapper**

**def** mapper(key, value): words=key.split() **for** word **in** words:

Wmr.emit(word, ’1’)

**3.4.2 reducer**

**def** reducer(key, iter):

sum = 0

**for** s **in** iter:

sum = sum + int(s) Wmr.emit(key, str(sum))

**Note:** Notes for this WMR interface for Python3 (see spec for details):

• The mapper and reducer for this interface are functions, as was the case for Scheme.

• Python provides many useful string manipulation methods for string objects, as well as convenient type conver- sion functions int() and str().

• The reducer uses a Python-style iterator, which may be used conveniently in a for loop construct.

**3.5 Comparison**

For comparison, here is an implementation of word count mapper and reducer for Java using Hadoop map-reduce directly, without using WMR.

*// Java WordCount for Hadoop*

*// Based on Hadoop documentation*

**package** wc;

**import java.io.IOException**;

**import java.util.\***;

**import org.apache.hadoop.fs.Path**; **import org.apache.hadoop.conf.\***; **import org.apache.hadoop.io.\***; **import org.apache.hadoop.mapred.\***; **import org.apache.hadoop.util.\***;

**public class WordCount** {

**public static void** main(String[] args) **throws** Exception { JobConf conf = **new** JobConf(WordCount.class); conf.setJobName("WordCount");

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

conf.setMapperClass(Map.class); conf.setCombinerClass(Reduce.class); conf.setReducerClass(Reduce.class);

conf.setInputFormat(TextInputFormat.class);

conf.setOutputFormat(TextOutputFormat.class);

FileInputFormat.setInputPaths(conf, **new** Path(args[0])); FileOutputFormat.setOutputPath(conf, **new** Path(args[1]));

JobClient.runJob(conf);

}

**public static class Map extends** MapReduceBase

**implements** Mapper {

**private final static** IntWritable one = **new** IntWritable(1);

**private** Text word = **new** Text();

**public void** map(LongWritable key, Text value, OutputCollector output,

Reporter reporter) **throws** IOException { String line = value.toString();

StringTokenizer tokenizer = **new** StringTokenizer(line);

**while** (tokenizer.hasMoreTokens()) { word.set(tokenizer.nextToken()); output.collect(word, one);

}

}

}

**public static class Reduce extends** MapReduceBase

**implements** Reducer {

**public void** reduce(Text key, Iterator values, OutputCollector output, Reporter reporter) **throws int** sum = 0;

**while** (values.hasNext()) {

sum += values.next().get();

}

output.collect(key, **new** IntWritable(sum));

}

}

}

**CHAPTER**

**FOUR**

**ACCESSING CONCURRENCY AND**

**PARALLELISM WITHIN A PROGRAMMING LANGUAGE**

**4.1 Features that are part of a programming language**

• **Java** synchronized: a method or a segment of code within a method can be marked as synchronized, meaning that no two threads of execution may execute in synchronized sections for the same object at the same time.

• **Ada** (developed for DOD applications in 1980s): threads (“concurrent tasks”), which may be created dynami- cally; “rendezvous” (cf. remote procedure call) with synchronized communication.

• **Erlang**: The language consists of (functional programming) *sequential* constructs plus additional *concurrent* constructs for carrying out sequential code in parallel. No threads, just processes – a design decision to allow for easier fault tolerance, because shared resources such as memory are very difficult to manage correctly in the presence of faults.

**4.2 Libraries**

• **MPI (Message Passing Interface)**: Library allowing for send, receive, and other communication calls for both *point-to-point* and *collective* communication in a distributed system. Supports a notion of “communicator groups” of processes.

• **Java** Thread **class**; java.util.concurrent: These are standard packages in the Java language. The Thread class may be subclassed to provide a (sequential) run() method for carrying out specified code when that thread object is started. The package java.util.concurrent provides programming interfaces and classes for concurrency-safe data structures, thread management/reuse and scheduling, synchronization primitives such as semaphores, etc.

**4.3 Other approaches**

• **Operating system calls**: For example, Linux provides fork() for creating new processes, socket() and related system calls for creating communication lines between processes that may be on separate machines, read() and write() for sending and receiving along socket connections, select() for synchronizing communication, and various thread packages exist for Linux.

**Map-Reduce in Various Programming Languages,**

• **OpenMP C/C++ pragmas**: A convenient approach for incrementally adding synchronization in a shared- memory multiprocessor (such as a multicore system), in which one adds preprocessor pragmas to request parallelization of a for loop, creation of threads, etc.

• **CUDA programming of a GPGPU**: Modern video controllers are highly parallel devices designed for highly parallel, very fast linear algebra computations that feed a pipeline for adding further graphics features (such as texturing). NVIDIA and other manufacturers now provide a programming interface enabling a programmer to make general-purpose computations with that specialized hardware—“General Purpose Graphics Processing Unit (GPGPU)”. NVIDIA’s CUDA language provides a library for C and C++ programs for the CPU that interact with separate “kernel” programs written for the GPU in order to perform such GPGPU computations.