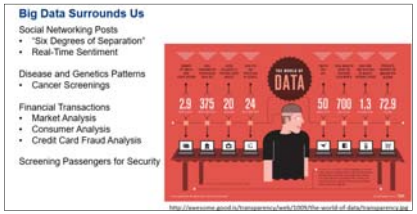
Using Big Data for Learning about a Slice of Parallel Computation in Several Courses

Bruce W. Char, William M. Mongan, Jeffrey L. Popyack
Drexel University

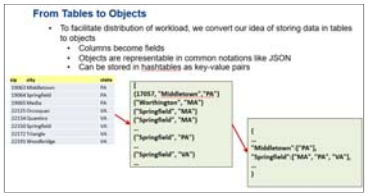
Overview

- Big Data parallelism is integrated into the undergraduate core curriculum in several modules within existing required courses.



Background

- Revamp existing core courses with parallelism curriculum, including
  - CI101 Computing and Informatics Design I, a first-term course taken by all CCI freshmen, including Computer Science, Software Engineering, Information Science, Informatics, Computing Security Technology, and Information Technology students
  - CS283 Systems Programming, a mid-level core course
  - CS385 Evolutionary Computing, an advanced elective course in Artificial Intelligence where machine-learning oriented big data analytics is possible
- All 250+ incoming freshmen receive instruction in CI101, while approximately the same number take CS283 each year. The Artificial Intelligence electives have approximately 30 students per offering of the course, approximately 1 to 2 times per year.
- We shift from the traditional "parallelism in the small" coverage of pthreads in CS283 to enable this coverage.



Educational Objectives

- Utilize MapReduce as a vehicle for exploring functional programming
- Work on problems that require computational efficiency and sound parallel design
- Introduce parallel computing within the context of relevant problems in big data and analytics, which will help prepare students for new and exciting co-op experiences
- Implement concepts in parallel computing throughout the core and advanced elective portions of the Computer Science curriculum, so that concepts can be properly introduced, reinforced, and then applied in sequence

Educational Design

- Present a lesson and lab on WebMapReduce MR-based SIMD parallelism in the freshman year in CI101
- Build upon the CI101 experience in CS283, working directly in Hadoop with supporting technologies
- Apply these concepts to a unique and non-trivial problem in an Artificial Intelligence course, using Spark and Mahout to make interesting queries against large data sets that require parallelism for efficient processing
- By incorporating parallelism into a first term course, we are able to present and apply elementary programming concepts just-in-time in the context of parallel computation
- We begin with "unplugged" non-computational examples and exercises to ease this transition, and then build upon those examples by writing parallelizable code

```
SEARCH_URL = "https://api.twitter.com/1.1/search/tweets.json?q=@SIGCSE_TS"

resp, data = oauth_req(SEARCH_URL, ACCESS_TOKEN, ACCESS_SECRET)

io = json.loads(data)

print data     # print the raw json returned

io = io['statuses']

for x in io:
    print x['created_at'] + ": " + x['user']['screen_name'] + ' said ' + x['text']
```

```
function Mapper(jsmr_context, data) {
  var words_list = data.split(' ');
  var dates_map = {};

  var the_date = words_list[0] + ' ' + words_list[1] + ' ' + words_list[2];

  if (the_date in dates_map) {
    dates_map[the_date]++;
  } else {
    dates_map[the_date] = 1;
  }

  for (the_date in dates_map) {
    var count = dates_map[the_date];
    jsmr_context.Emit(the_date, count);
  }
}
```

```
function Reducer(jsmr_context, key) {
 // sum date counts from each line to get total for each date
 var total_count = 0;
 while (jsmr_context.HaveMoreValues()) {
  var value_str = jsmr_context.GetNextValue();
  total_count += parseInt(value_str);
 }
 jsmr_context.Emit(key + ':' + total_count.toString());
}
```

```
Sun Jan 19:3
Thu Jan 16:1
Tue Jan 14:1
Tue Jan 21:2
Wed Jan 15:4
Wed Jan 22:2
```

Implementation: CI101

- Brief "unplugged" activity in which students count the number of students in their row, and aggregate those counts to subsequent rows to efficiently count the number of students in attendance.
- Discuss concept of SIMD as a paradigm for achieving parallelism.
- Using WebMapReduce "Word Count" example, step through a similar example and identify the concept of a mapper and a reducer, as well as a key and a value mapping.
- In lab, students perform another "unplugged" activity in which they identify key/value mappings and a strategy for counting the number of each suit in a deck of cards.
- Then, students explore several examples in WebMapReduce, some developed by the authors of WebMapReduce, and others developed in-house, including a GPA calculator.
- Finally, students develop their own MapReduce task using WebMapReduce to aggregate graduation rate data from raw School District of Philadelphia data obtained from OpenDataPhilly.

```
{'Alice':95, 'Bob':66, 'Carol':57, 'Dave':81, 'Eve':86}
{'Eve':55, 'Alice':87, 'Bob':82, 'Carol':74, 'Dave':74}
{'Dave':79, 'Bob':56, 'Carol':70, 'Eve':50, 'Alice':63}
```

```
def mapper(key, value):
    grade_map = eval(key) # automatically parses the data in JSON format
    # grade_map['Alice'] contains 95 87 or 63 (and so on) depending on which row the mapper is reading

    for student in grade_map: # student is 'Alice', 'Bob', and so on
        # get each of the 4 grades in your mapper data line
        grade = grade_map[student]
        Wmr.emit(student, grade)
```

```
def reducer(key, values):
    sum = 0
    count = 0
    for value in values:
        sum = sum + float(value)
        count = count + 1
    if count > 0:
        average = sum / count
    Wmr.emit(key, average)
```

Question 2: What does each mapper use as its "piece" of the data? For example, is it a word, a paragraph, the whole poem, *etc.*?

Question 3: It seems surprising that this program calls so many mappers, each with a small part of the data. Why do you think MapReduce jobs are executed this way? For a small poem such as the one in this example, it would seem faster to just run the function on the whole poem at once. Why do you think this would run faster as the input becomes larger?

Question 4: Modify the line of code
var words_list = data.split(' ');
by removing the space inside the parentheses, and run your MapReduce job again (you may have to click Reset in between runs). What happens?

Question 5: Looking at the mapper function, how do you think the function takes its piece of the data, and produces its emitted values (*i.e.*, "Hope: 1")? Describe what the code in the mapper function does, step-by-step.

Question 6: What does each reducer use as its input, and what does each reducer emit? See the Debug Log to the right when you run your MapReduce job for clues.

Evaluation Plan

Pre- and Post- Surveys are administered to students before and after each module in the sequence, measuring sentiment and also understanding of foundational concepts and their applications.

Aggregate performance in downstream courses can be compared against historical data.

Dissemination and Future Work

- Student performance was favorable in CI101.
- Students pointed out that the lab was challenging.
- Expand upon the MapReduce tasks that students build, and the guided exploratory questions that they answer, so as to better manage the challenge and learning curve associated with presenting this to freshmen students.
- Utilize larger data sets early, so as not to merely introduce the concepts but, in fact, mandate their use and appreciate their speedup early on.

References and Acknowledgments

RAGUE, B. Measuring CS1 perceptions of parallelism. In Frontiers in Education Conference (FIE), 2011 (2011), IEEE, pp. S3E–1.

The authors thank Mike Fenton for his work in designing laboratory exercises and questions along with the instructional team.

The authors would like to thank Dick Brown and his CSInParallel colleagues for help with WebMapReduce, and Sushil Prasad and the staff of TCPP/CDER Center for their assistance in becoming familiar with the materials available through them.

This work is supported in part by a TCPP CDER Center Early Adopter Award.

This material is based upon work supported by the National Science Foundation under Grant No. CNS-1301171.
Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Supported in part by IBM Big Data Faculty Awards 2013

Contact Us

Bruce W. Char, Professor of Computer Science: charbw@drexel.edu

William M. Mongan, Associate Teaching Professor of Computer Science and Associate Department Head for Undergraduate Affairs: wmm24@drexel.edu

Jeffrey L. Popyack, Professor of Computer Science: popyack@drexel.edu