Introduction

CPS 296.1
Database and Programming Languages: Crossing the Chasm

Jun Yang
Duke University
January 14, 2010

† Thanks to contents/ideas borrowed from
Babu (http://www.cs.duke.edu/courses/fall09/cps216/),
Cook (http://www.cs.utexas.edu/users/wcook/Courses/PLDB2003/),
and Loo (http://www.cis.upenn.edu/~boonloo/research/talks/dimacs-dbpl.pdf)

Let’s start with a poll

- What percentage of the programs you wrote interfaced with a database system (DBMS)?
- What percentage of the programs you wrote interfaced with some persistent store (e.g., file system)?
- How about real-world applications out there?
  - ➲We’ve got a widespread problem
- How often did you run into the following situation?
  - “It would be cool if I let a DBMS manage data for this program”
  - “Forget it… it’s such a hassle to get data in and out”
  - ➲We’ve got pain

➲Opportunity for high-impact research

Beyond persistence

- Almost all systems require persistence
- Database systems (DBMS) is one paradigm for persistence
- Are there other DBMS features that make it attractive to use?
  - Declarative (high-level) access
    - Most often: SQL over relational data
    - There are others data models and query languages too
  - Run-time, data-driven optimization
    - + persistence $\rightarrow$ good performance for large data
  - Transactions
    - Atomicity + Consistency + Isolation + Durability

➲So our problem is more general than adding persistence to programming languages

Another poll

- How many programming languages and paradigms have you heard about or used over the years?
- Would you ever commit to one programming paradigm?

- One size doesn’t fit all—different tasks (sometime within one application) call for different programming paradigms

➲General problem: interfacing components with different programming paradigms together in one application

Why is it hard?

- Impedance mismatch
  - E.g., between DB and PL:
    - Flat tables vs. complex objects
    - Declarative queries vs. procedural programs
    - Transactions vs. semaphores
  - Leads to not only lots of tedious code, but also inefficient data repackaging
- Lack of automatic optimization across components
  - Programmer decides what computation/data go across
  - Needs expertise and tuning
  - May result in suboptimal communication

Why is it hard?

- Cultural mismatch
  - DB researchers don’t understand PL research
    - “They are working on toy problems that fit in memory”
  - PL researchers don’t understand DB research
    - “They are working in a narrow domain and with an ugly language”
  - And anyway, it is not our problem
    - “Industry and users will figure it out”

A closer look…

… at the PL/DB interaction in several app domains:

- Web-based client/server apps
- Scientific/statistical computing
- Massive data analytics
- System building

Image by Volker Hartmann/AFP/Getty Images

7

Web-based client/server apps

How do you like the client (pseudo) code below?

```
conn = new DBConnection(connURL, user, passwd);
resultSet = conn.query("SELECT PID, NAME, DESCRIPTION FROM Product");
while (resultSet.next()) {
  pid = resultSet.getInt(1);
  name = resultSet.getString(2);
  description = resultSet.getString(3);
  resultSet2 = conn.query
    ("SELECT * FROM Order WHERE PID = " + pid.toString());
  count = 0; while (resultSet2.next()) count++;
  resultSet2.close();
  productList.append(new Product(pid, name, description, count));
}
resultSet.close();
conn.close();
```

- Manual schema mapping and data repackaging
- Explicit boundary between PL/DB processing dictated by code
  - Suboptimal, but perhaps easier to write for some programmers

8

Scientific/statistical computing

How do you like the (pseudo) code snippet below on big datasets?

```
V1 = vector_from_db(DB, "SELECT I, VALUE FROM VECTOR1");
M1 = matrix_from_db(DB, "SELECT I, J, VALUE FROM MATRIX1");
% similarly load M2, M3, V2...
V3 = (std(V1)*M1 + M2) * M3 * V2;
```

- Chances are the code will thrash and perform horribly
  - I.e., it uses up physical memory and OS starts swapping to/from disk
  - Will I/O-efficient operator (+, *) libraries alone solve the problem?
- Which vectors/matrices can we avoid loading into memory?

➲ Again, boundary between PL/DB processing dictated by code

- Can multiplications be reordered automatically?
- DB storage/layout of vectors/matrices may or may not be efficient

9

Massive data analytics

- Massive data requiring thousands of machines to process
  - Numbers from Google in 2007:
    - Data processed per month is 400 PB (PetaBytes)
    - Average job size is 180 GB: half an hour just to read it from disk (@100 MB/s), or 10 hours to download (@5 MB/s)
- Relatively simple computation (e.g., extraction, filtering, grouping, aggregation) for now, but this can change
- PL is evolving: restrictive programming models like MapReduce
  - Not so much to maximize performance, but to
    - Make programs simpler to parallelize automatically
    - Relieve programmers from worrying about fault tolerance and load balancing
- People quickly jumped on lack of DBMS features and started adding them—a new breed of scalable "DBMS" has emerged
  - But how will they interface with PL?
  - Can we get it right this time?

10

System building

- Systems are becoming more complex and harder to build
  - E.g., the Chord DHT, a P2P network, takes 10,000 LOC
  - Not only a pain to write, but also a pain to verify
- Datalog: declarative query language with recursion studied in depth by the DB community
  - Chord in a Datalog-like language: 48 lines, easy to verify
- What about efficiency?
  - No fundamental overhead
  - Can benefit from well-studied Datalog optimization techniques

  ➲ Same question asked of relational DBMS in the 70's
- Other tasks: trust management, network monitoring
  - Used to be different subsystems speaking different languages
  - Datalog provides hope to unify them

11

Summary of challenges

- For some applications, we might be able to make the chasm disappear to programmers
  - How do we design one unifying programming paradigm?

But if the chasm stays…

- How can we automate mapping across the chasm?
  - Between data, language constructs, and semantics
- How can we optimize across the chasm?
  - Programmers can place data and computation wherever they like, but we still can move data and computation across the chasm to improve performance
  - What to do on one side of the chasm may depend on what happens on the other side

12

Adding to the challenges…

There is more pain now than ever

- Growing data volume
- Growing system complexity
- Proliferation of data models, languages, paradigms, systems specialized in different tasks

➲Inefficiency gets compounded, optimization becomes harder, and programmers are overwhelmed and increasingly "picky" (specialized)

13

Goal of this course

- Find a way to cross the chasm or to make it disappear, by
  - Studying techniques and examples
  - Casting away our PL/DB prejudices
  - Looking to past, present, and future

Image from http://www.ibm.com/developerworks/rational/library/4620.html

14

Course information

- Format: seminar + project
  - No textbook
  - Read, review, and discuss lots of papers
  - Do a project (1-3 persons per team)
- Web: http://www.cs.duke.edu/courses/spring10/cps296.1/
  - Your portal to everything 296.1: slides, schedule, reading assignments, review submission, etc.
- No mailing list (to hide from spam)
  - Reply-all to a message that I send to the whole class allows you to contact everybody
- Time/location: TTH 2:50-4:05pm, North 306
- Office hours: TTH 1:30-2:50pm, or by appointment

15

Course load and grading

- Reading, discussion, and participation (50%)
  - Short reviews for reading assignments (20%)
    - No more than 4 papers per week
  - Present and lead discussions in teams of 2 to 3, 2-3 times (20%)
  - Attendance (10%)
- Course project (50%)
  - Work on something that you'll be proud of
  - Proposal presentation in class (15%) after spring break
  - Short progress report (5%) on April 8
  - Final presentation in class + final report (30%) in the final slot

➲More details on course website

16

Course roadmap (tentative)

- Weeks 1-3: historical perspectives
  - Data models and query languages
  - Object-oriented and object-relational DBMS
  - Persistent PL
- Weeks 4-5: recent advances in client/server settings
  - Object-relational mapping, language-integrated query, and how to optimize them
  - Tour of Hibernate, Django, and LINQ
- Week 6: massive data processing
  - Dryad/LINQ, SCOPE
- Week 7: scientific/statistical computing
- ½ of Week 8: catch-up or topic of choice—open to suggestions
  - Parallel data processing by GPU programming?

17

Course roadmap (cont'd)

- Week 9: spring break
- Week 10: project proposal talks
- Week 11 + ½ of Week 12: compilation essentials
- ½ of Week 12 + Week 13: more scientific/statistical computing
  - End of Week 13: short project progress report
- Week 14: more massive data processing
  - Pig and ScalaQL
- Week 15: declarative networking and access control
- Week 16: graduate reading period
- Week 17: show time!

18

Course rules

- Think deeply
- Show respect
- Question authority
- Be creative
- Commit to Honor Code

➲You must be self-motivated in this class

Image from somewhere on digg.com

19

Reading for next week

3 papers by Stonebraker et al.

- Historical perspectives on data models and query languages
  - No review due
- Postgres, *the* DBMS that started the "object-relational" era
  - No review due
- And how Postgres added user-defined types
  - Review!

➲See course website for links to PDFs, what to write in a review, how to submit, and tips on reading

20