Group project Specification --- Shared Notes

Team 48
Zhengyu Sun, Ziyi Yan, Lyuliang Liu

1. Proposal

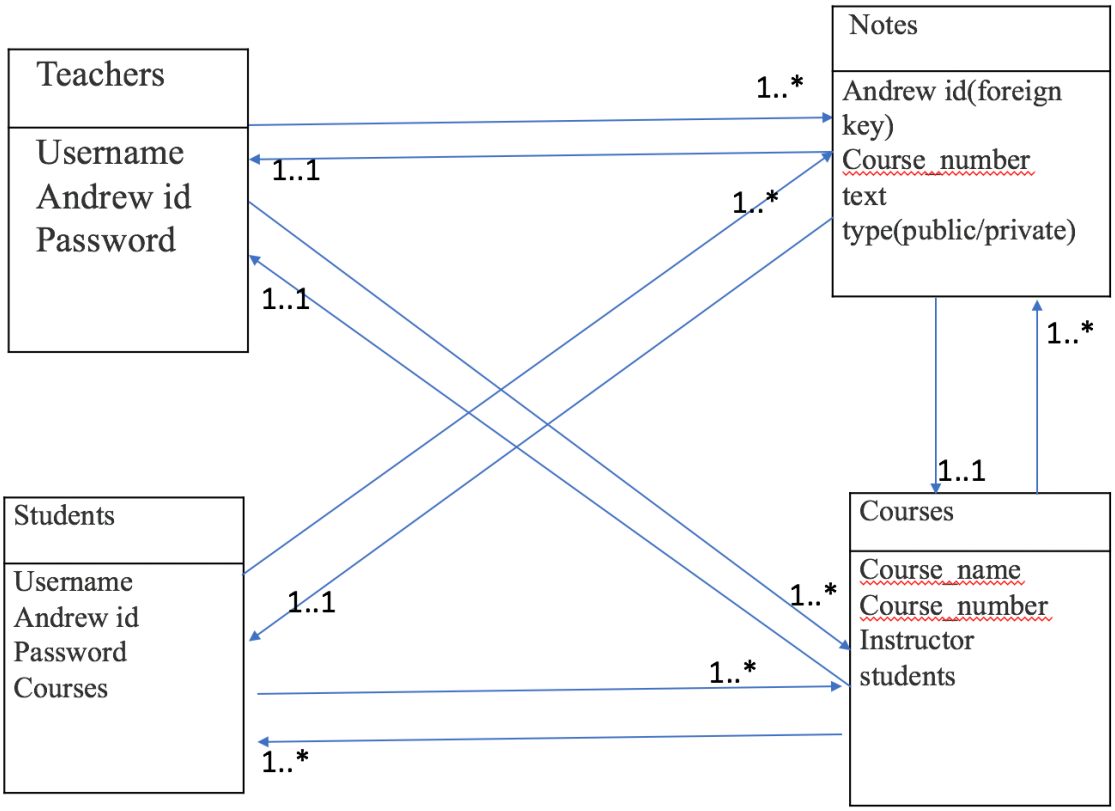
Nowadays, we notice that in order to avoid missing any important points, many students prefer to take notes in lectures, highlight important sentences on slides as well as the homework write-ups. Instructors also prefer to highlight important knowledge points or instructions in homework write-up for students to pay enough attention to. Besides, during the group projects, students often need to share their works as well as notes with each other, which can help them complete projects efficiently. So in order to help students improve their study, we decide to implement a web-based application that can provide a more efficient way to allow students share notes and materials about different courses. This application is designed for all courses.

First, the accounts will be divided into 'students' and 'instructors' when users try to register for the application. Instructors can register different courses on our application when they log in and all courses can be enrolled by students. In the course profile, instructors can upload the course slides and also add annotations on slides shared with students who enrolled in this course. Instructors can also see all notes shared or uploaded by students in this course as well as adding new notes online.

When students enroll in a course, he or she can add new notes, choose to share notes with the whole class, see others notes in this course as well as adding new annotations on slides or other pdf materials. Besides, students can filter notes to see notes from instructors or someone.

All notes that are displayed on the public area of the course can be marked as 'favorite' by students and be stored in the student's own repository.

2. data models



3. Functionalities

Module 1: Authentication

Responsible person: Zhengyu Sun

1. Register and Login
   Users can register as teachers or students. Then they can login in with different user permissions.

Module 2: Course-related operation

Responsible person: Zhengyu Sun

2. Create courses
   Teachers can create a course. Later the students who joined this course can enter the course page, access the public files and make their notes in the course.

3. Search a course
   Students can search a course that they want to join with the course id, using a search box.

4. Join a course
   Students can join a course. Then they can enter the course page to see the read material uploaded by the teacher, and public notes created or uploaded by their classmates, as well as make their own notes.

Module 3: File manipulation

Responsible person: Ziyi Yan

5. Upload files
   Teachers can upload course-related reading materials to the course page, which are accessible to all students who joined this course. If students made notes on their local machines, they can also upload the notes to their course page in order to store and organize their notes. They can choose to make it public or keep it private. If a created note is public, all the students who have joined this course can see this note on their course page, and are able to download it. If a created note is private, only the creator herself can see this note on her course page.

6. Download files
   Users can download the reading materials and available notes from the course page. Available notes including the notes of the student herself created, or the notes other students created and set to be public.

7. Favorite Notes
   Student can add a note into their Favorite Notes, which will appear on their homepage.

Module 4: .pdf type notes manipulation

Responsible person: Lyuliang Liu

8. Make annotations
   Students can make annotations on the reading material that the teacher uploaded, such as drawing some lines or write some words on it. Once a student has made some annotations, a copy of the reading material containing the notes will be created and appear on the student's course page. The student can choose to make it public or keep it private. If the note is public, all the students who have joined this course can see this note on their course page, and are able to download it. If a note is private, only the creator herself can see this note on her course page.

   Module 5: .docx type notes manipulation
   Responsible person: Lyuliang Liu

9. Create notes
   Students can create a new note for a course. It works like an online document editor. Created notes will also appear on the student's course page. After saving the note, they can choose to make it public or keep it private. If a created note is public, all the students who have joined this course can see this note on their course page, and are able to download it. If a created note is private, only the creator herself can see this note on her course page.

10. Edit notes
    After creating a note, the student can open it and edit it.

11. Share editable notes (Optional)
    Students can share the editing permission of a note they created with other students in this course. If a student get permission to edit a note created by another student, the note will appear on her course page, and she can edit it. Her editing result will reflect in the same note of every student who are sharing this note.

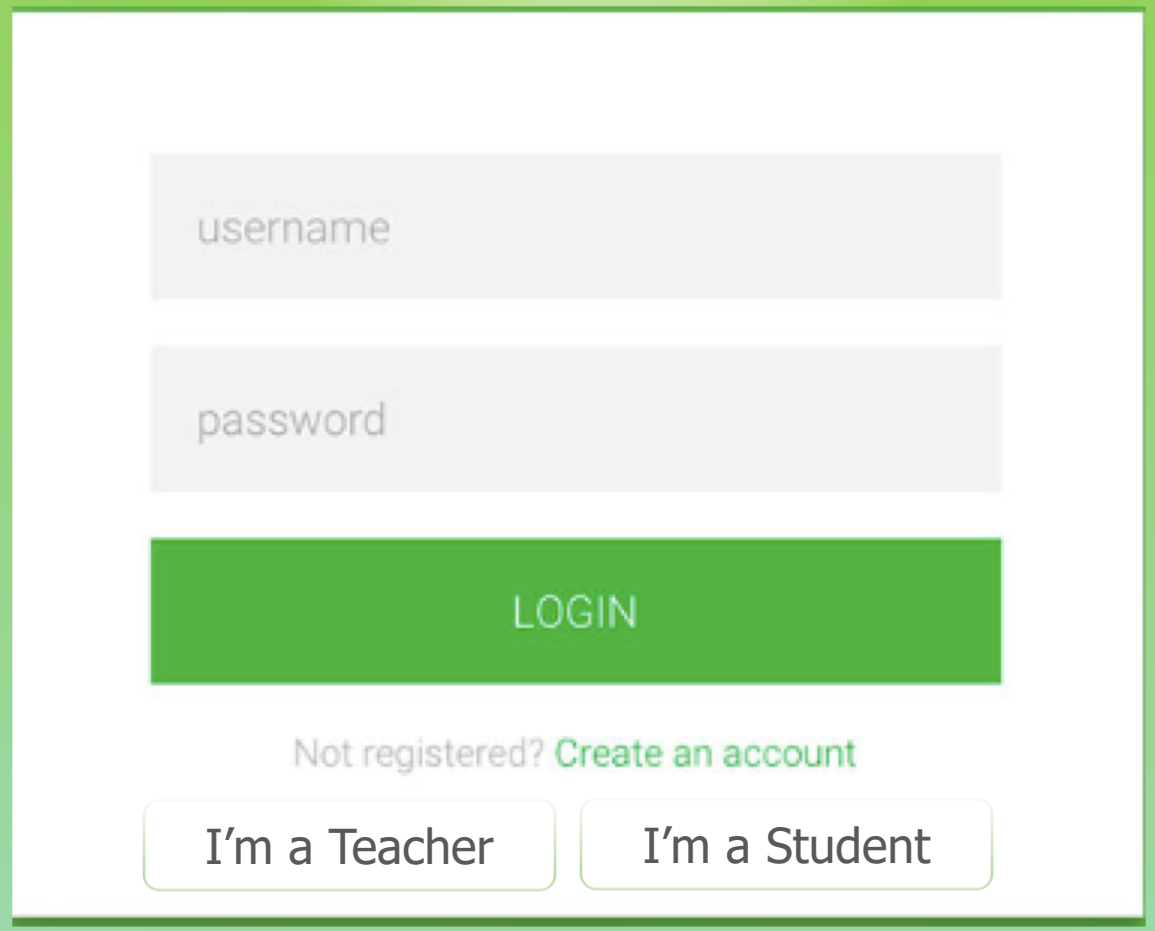
4. Wireframes

Shared Notes

Create Your Own Notes
Enjoy Your Study
Be happy every day
Have Fun

Login / Register

Shared Notes



Shared Notes

Trace-based Just-in-Time Type Specialization for Dynamic Languages

Andreas Gal*+, Brendan Eich*, Mike Shaver*, David Anderson*, David Mandelin*,
Mohammad R. Haghighat$, Blake Kaplan*, Graydon Hoare*, Boris Zbarsky*, Jason Orendorff*,
Jesse Ruderman*, Edwin Smith#, Rick Reitmaier#, Michael Bebenita+, Mason Chang+#, Michael Franz+

Mozilla Corporation*
{gal,brendan,shaver,danderson,dmandelin,mrbkap,graydon,bz,jorendorff,jruderman}@mozilla.com

Adobe Corporation#
{edwsmith,rreitmai}@adobe.com

Intel Corporation$
{mohammad.r.haghighat}@intel.com

University of California, Irvine+
{mbebenit,changm,franz}@uci.edu

Abstract

Dynamic languages such as JavaScript are more difficult to compile than statically typed ones. Since no concrete type information is available, traditional compilers need to emit generic code that can handle all possible type combinations at runtime. We present an alternative compilation technique for dynamically typed languages that identifies frequently executed loop traces at run-time and then generates machine code on the fly that is specialized for the actual dynamic types occurring on each path through the loop. Our method provides cheap inter-procedural type specialization, and an elegant and efficient way of incrementally compiling lazily discovered alternative paths through nested loops. We have implemented a dynamic compiler for JavaScript based on our technique and we have measured speedups of 10x and more for certain benchmark programs.

Categories and Subject Descriptors D.3.4 [*Programming Languages*]: Processors — *Incremental compilers, code generation.*

General Terms Design, Experimentation, Measurement, Performance.

Keywords JavaScript, just-in-time compilation, trace trees.

1. Introduction

Dynamic languages such as JavaScript, Python, and Ruby, are popular since they are expressive, accessible to non-experts, and make deployment as easy as distributing a source file. They are used for small scripts as well as for complex applications. JavaScript, for example, is the de facto standard for client-side web programming and is used for the application logic of browser-based productivity applications such as Google Mail, Google Docs and Zimbra Collaboration Suite. In this domain, in order to provide a fluid user experience and enable a new generation of applications, virtual machines must provide a low startup time and high performance.

Compilers for statically typed languages rely on type information to generate efficient machine code. In a dynamically typed programming language such as JavaScript, the types of expressions may vary at runtime. This means that the compiler can no longer easily transform operations into machine instructions that operate on one specific type. Without exact type information, the compiler must emit slower generalized machine code that can deal with all potential type combinations. While compile-time static type inference might be able to gather type information to generate optimized machine code, traditional static analysis is very expensive and hence not well suited for the highly interactive environment of a web browser.

We present a trace-based compilation technique for dynamic languages that reconciles speed of compilation with excellent performance of the generated machine code. Our system uses a mixed-mode execution approach: the system starts running JavaScript in a fast-starting bytecode interpreter. As the program runs, the system identifies *hot* (frequently executed) bytecode sequences, records them, and compiles them to fast native code. We call such a sequence of instructions a *trace*.

Unlike method-based dynamic compilers, our dynamic compiler operates at the granularity of individual loops. This design choice is based on the expectation that programs spend most of their time in hot loops. Even in dynamically typed languages, we expect hot loops to be mostly *type-stable*, meaning that the types of values are invariant. (12) For example, we would expect loop counters that start as integers to remain integers for all iterations. When both of these expectations hold, a trace-based compiler can cover the program execution with a small number of type-specialized, efficiently compiled traces.

Each compiled trace covers one path through the program with one mapping of values to types. When the VM executes a compiled trace, it cannot guarantee that the same path will be followed or that the same types will occur in subsequent loop iterations.

PLDI'09, June 15–20, 2009, Dublin, Ireland.

Hence, recording and compiling a trace speculates that the path and typing will be exactly as they were during recording for subsequent iterations of the loop.

Every compiled trace contains all the *guards* (checks) required to validate the speculation. If one of the guards fails (if control flow is different, or a value of a different type is generated), the trace exits. If an exit becomes hot, the VM can record a *branch trace* starting at the exit to cover the new path. In this way, the VM records a *trace tree* covering all the hot paths through the loop.

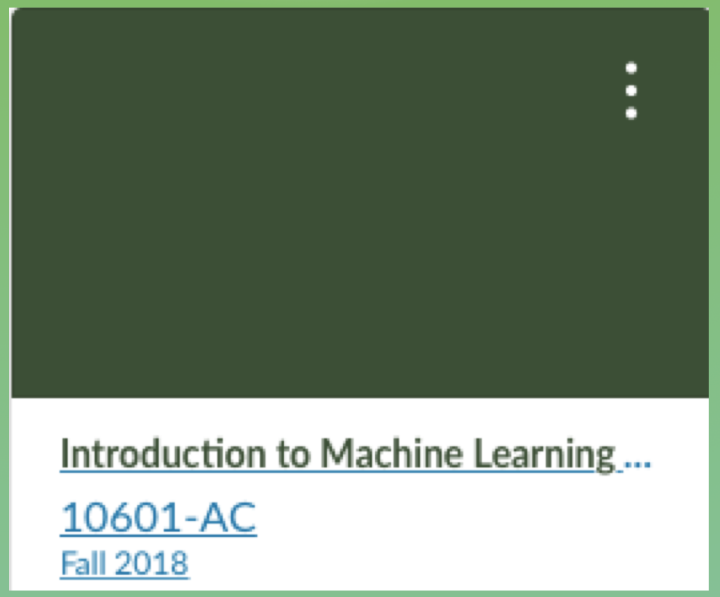
Nested loops can be difficult to optimize for tracing VMs. In a naive implementation, inner loops would become hot first, and the VM would start tracing there. When the inner loop exits, the VM would detect that a different branch was taken. The VM would

```
1 for (var i = 2; i < 100; ++i) {
2   if (!primes[i])
3     continue;
4   for (var k = i + i; k < 100; k += i)
5     primes[k] = false;
6 }
```

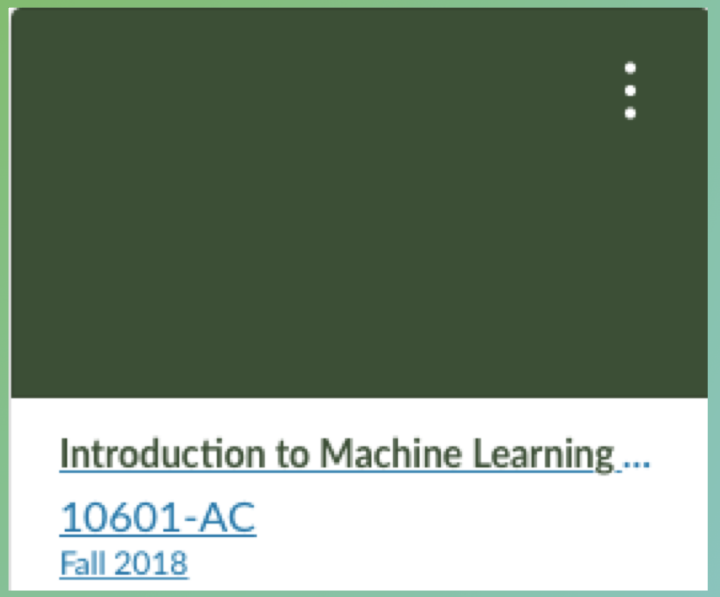
Figure 1. Sample program: sieve of Eratosthenes. primes is initialized to an array of 100 false values on entry to this code snippet.

Shared Notes

View Notes



View Notes



View Notes

Shared Notes

Edit

Bookmark

Edit

Bookmark

Edit

Bookmark

Create A New Post