Dyninst: An API for Runtime Code Patching

presented by

Skylar Byrd Rampersaud

http://www.cs.umd.edu/~byrd/dyninst.ppt

The First Slide

- Goal: change a program while it is executing
  - Without recompiling, relinking or restarting
- Applications
  - Dynamic performance measurement
  - Performance steering in large-scale simulations

Process Model

- A program can attach to a running program
- Create a new bit of code
- Insert it into the program
- Can augment or change subroutines

Dyninst is Not

- An instrumenting compiler
- Adding binary code to an executable before it is run
- Machine code (assembly language)

Terminology

- Point - a location where code can be inserted
- Snippet – representation of executable code to be inserted
- Thread – thread of execution
- Image – the static on-disk program

Abstractions

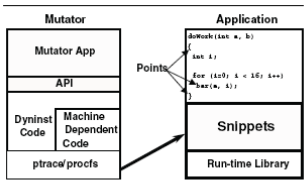

Fig. 1 Abstractions used in the API

3 Main Interface Components

- Classes to manipulate executing code
  - BPatch, BPatch_thread
- Classes to access the original image and data structures
  - BPatch_module, BPatch_function
- Classes to construct and insert new code snippets
  - BPatch_point, BPatch_snippet

Statements to be Added

- A collection of BPatch_snippet instances (and subclasses representing specific types of code)
  - Collection forms a direct acyclic graph
  - Abstract Syntax Tree created from leaf to root

Types

- The API includes a simple type system
  - Integers, strings, floats
  - Support for aggregate types

Events

- API provides notification of application events
- Also provides a way to query for specific events

How Does It Work?

- Mutator process uses debugger-style OS functions to access memory and events of running process
- Translate snippets into machine code
- Copy code into an array in the running process
- Uses “trampolines” to for transferring execution to inserted code

Trampolines

- Replace some instructions with a branch to a base trampoline
- Base trampoline branches to a mini-trampoline
- Base trampoline executes the original instructions once execution returns from the mini

Mini-trampoline

- Saves registers and other state
- Contains code for one snippet
- Can chain these together to include multiple snippets at one point
- Branches back to the base trampoline at the end of the final snippet

Trampolines Illustrated

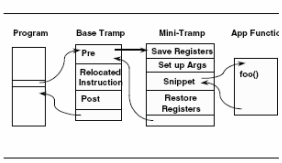

Fig. 2 Inserting code into a running program

Three Example Programs

- Procedure call counting
- RETEE
- Conditional breakpoints

Procedure Call Counting

- Mutator creates an instance of the BPatch class
- Identifies process (running or not)
  - Creates new thread or new process
- Defines snippets and points
  - Instrumenting a single function may require multiple points
- Creates a new variable in the target space

Counting Procedure Calls

```
1  BPatch bpatch;
2  BPatch_thread *appThread=bpatch.createProcess(pathname, argv);
3  BPatch_image *appImage = appThread->getImage();
4  BPatch_Vector <BPatch_point*> *points =
5      appImage->findProcedurePoint("InterestingProcedure", BPatch_entry);
6  BPatch_variableExpr *intCounter =
7      appThread->malloc(*appImage->findType("int"));
8  BPatch_arithExpr addOne(BPatch_assign, *intCounter,
9      BPatch_arithExpr(BPatch_plus, *intCounter, BPatch_constExpr(1)));
10 appThread->insertBlock(addOne, *points);
```

Fig. 3 Code to count the number of occurrences of "InterestingProcedure"

RETEE

- Uses the one-time code feature of the API

```
1  BPatch_function *openFunc = appImage->findFunction("open");
2  BPatch_Vector<BPatch_snippet *> openArgs;
3  BPatch_constExpr fileName(argv[3]);
4  openArgs.push_back(&fileName);
5  BPatch_constExpr fileFlags(O_WRONLY|O_CREAT);
6  openArgs.push_back(&fileFlags);
7  BPatch_constExpr fileMode(0666);
8  openArgs.push_back(&fileMode);
9  BPatch_funcCallExpr openCall(*openFunc, openArgs);
10 BPatch_variableExpr *fdVar =
11   appThread->malloc(*appImage->findType("int"));
12 BPatch_arithExpr openExpr(BPatch_assign, *fdVar, openCall);
13 appThread->oneTimeCode(openExpr);
```

Fig. 4 Code to open the log file in the application

Conditional Breakpoints

- Very slow in a traditional debugger
- Results averaged over 20 runs of the program

Table 1
Conditional Breakpoint Performance

	Breakpoints		Dyninst	gdb
Application	Number of Operations	Operations/ Second	Time (seconds)	Time (seconds)
compress95	32,513	406,655.7	0.08	74.35
li (xlmatch)	110,209	43,607.7	2.53	221.04
li (compare)	4475	640.2	6.99	16.39
li (binary)	401	19.4	20.69	21.62

Other Applications

- Online critical path analysis in SMPs
- Harmony
  - Use runtime observations to automatically tune programs
- Eliminate redundant synchronization in parallel programs
- Other debugging and performance monitoring tools

Related Work

- Binary editing tools
- ‘C
  - Allows a program to define a set of C-like statements and call them
- Instrumenting compilers
- Los Alamos Debugger

Conclusion

- Dyninst is a simple runtime API to allow creation and patching of programs
- Ability to create portable tools by providing machine-independent abstractions
- Implemented Platforms
  - Intel x86, Sun Sparc, Compaq Alpha, MIPS, IBM Power
- http://www.dyninst.org/