Compressing Java Class files

1999 ACM SIGPLAN Conference on Programming Language Design and Implementation

William Pugh
Dept. of Computer Science
Univ. of Maryland

Java Class files

- Compiled Java source programs generates (lots of) class files
- Architecture neutral
- Standard distribution format
  - Java programs can be compiled to native executables, but I won't talk about that

Class file contents

- class files contain lots of symbolic information
  - For javac, only 21% of uncompressed class file is bytecode
- Information for linking
- Allows code compiled against an old library
  - to be linked with a new library
  - so long as dependent functionality still there

Java **AR**chive (jar) files

- A collection of of class files and other resources (e.g., images)
- same format as zip archives
  - individual files can be compressed with zlib
- includes manifest
  - Information such as code signatures

Executing Java programs over the net

- Download and install program
  - compact archive format
- Execute as downloaded
  - class files as needed, or
  - compact and progressive archive format

Download individual class files as needed

- TCP set-up costs for each class file
  - unless you use persistent http connection
- No compression
- Get just the class files you need
  - some class files are needed only for verification
  - entire class file is needed if you need only one method
- This approach isn't used in practice
  - for non-trivial applications

Download jar archive

- 1 TCP connection
- zlib compression of individual class files
  - about a factor of 2 savings
- may download class files that are not used
- entire jar archive must be downloaded before any class files can be accessed

A Wire format for class files?

- Bandwidth most important
- Decompression time relatively important
  - Compression time not very important
- Progressive
- Not random access
- Can translate into jar archive or class files
  - or load directly into JVM

Debugging information

- Java class files often contain debugging information
  - source file
  - line number
  - local variables
- Will not include debugging information in wire format
  - could do so; would compress fairly well

Cleanup

- When comparing my format to jar files
  - Clean up first
- Remove debugging information
- Garbage collect constant pool
- Sort constant pool
  - improves compression
- Exclude non-class files from archive

Effects of Clean up

	j0r	jar	sjar	
	no	yes	yes	compressed?
	no	yes	no	debugging info?
	yes	no	yes	Cleaned up?
Hanoi	86	57	46	
icebrowserbean	226	125	116	
javafig_dashO	269	136	131	
javafig	357	198	170	
jmark20	309	189	173	
_213_javac	516	274	226	
ImageEditor	454	359	257	
tools	1,557	950	737	
visaj	2,189	1,524	1,157	
swingall	3,265	2,193	1,657	
rt	8,937	5,726	4,652	

Easy Java class file wire format: Collective Zip

- In standard Jar archive,
  - files are compressed individually
- gzip/zlib finds repeating patterns
- lots of patterns repeat between but not as much within class files
- Generate jar file without individual compression
- Compress the entire resulting jar file

Effectiveness of collective zip

Benchmark	Original Size	Collective Zip	Size of Collective Zip as % of Original
Hanoi	**46**	**31**	**67%**
IBM Host on demand	**98**	**85**	**87%**
ICE Browser	**105**	**88**	**84%**
JavaFig	**171**	**144**	**84%**
tools	**737**	**513**	**70%**
visaj	**1,157**	**703**	**61%**
swingall	**1,657**	**998**	**60%**
JDK 1.2 runtime	**4,652**	**2,820**	**61%**

A closer look at class file contents

	swingall	javac
Total size	3,265	516
excluding jar overhead	3,010	485
Field definitions	36	7
Method definitions	97	10
Code	768	114
Other	72	12
Constant pool	2,037	342
Utf8 entries	1,704	295
if shared	372	56
if shared and factored	235	26

What did we just learn?

- A substantial part of class files consists of constant pools
  - Bytecode is the only other substantial component
- Most of the space in constant pools is taken up by Utf8 entries
- Sharing Utf8 entries across class files is a huge win

Beating collective zip is hard

- A lot of the things you could do
  - e.g., share constant pool entries
- are already done by a collective zip
- You can work very hard
  - and find that you don't beat collective zip by much

Drawbacks to sharing

- Increases # of constant pool entries
  - How do we encode a reference?
- For most class files, less than 255 entries
  - can encode in a single byte

Compressing uniform streams

- class files are jumbles of different types
  - Utf8 encodings, bytecodes, constant pool entries
- Most compression algorithms work better if given a more uniform stream
  - separate out class files into streams for each type of information
  - compress each stream individually

Compressing bytestreams

- Zlib (and most compression algorithms) are designed to work on bytestreams
- How do you compress a stream of shorts?
  - Standard serialization mixes types
  - Could use separate streams for high and low bytes
  - Use variable length encoding
    - Hope that most entries can be encoded in a single byte

Encoding references

- How do you encode a reference to an object (e.g., a constant pool entry) you may have seen before?
  - so that most references are encoded in 1 byte
- Overload id's based on type
  - In almost all cases, know the type of the object being referenced

Encoding references (continued)

- Tried several schemes
- One that worked best was a move-to-front queue
  - Suggested by Ernst et al.
  - Long history in compression literature

Move to front queue

- Maintain a list of all the objects seen previously
- To encode an object seen previously
  - encode its position (1 for first entry)
  - move it to the front of the list
- To encode an object not seen previously
  - encode 0
  - put it at front of list

Implementation of Move-to-front queues

- Use a modified skip-list
  - links record distance they travel
- In decoder, a move-to-front operation on element *k* requires O(log *k*) time
  - regardless of total number of elements in list
- In encoder, requires O(log *n*) time

Factoring?

- The string "java.awt" occurs in the Utf8 encoding of many class names
- Method and field signatures contain separate Utf8 encoding of class names
- String f(String s) is recorded as having type (Ljava/lang/String;)Ljava/lang/String;
  - L is to differentiate between references and primitive types

Reorganize class file

- Factor information to avoid as much redundancy as possible
  - packageNames
  - simpleClassNames
  - classNames
  - method type (array of classnames)
  - ...

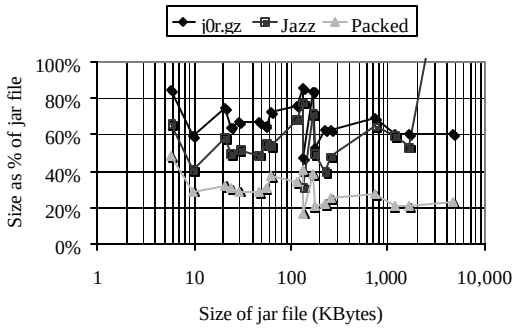
Compressing bytecodes

- Separating out opcodes from operands helps
- Use separate streams for :
  - opcodes
  - different register types
  - branch offsets
  - integer constants
  - constant pool references (already separate)

Compression results

Benchmark	jar size	collective zip	packed	Czip/ jar	packed/ jar
Hanoi	46	31	14	67%	30%
IBM Host on demand	98	85	44	87%	45%
ICE Browser	105	88	36	84%	35%
JavaFig	171	144	64	84%	37%
Cinderella	625	-	171		27%
tools	737	513	204	70%	28%
Lotus eSuite Sheet	1,101	-	549		50%
visaj	1,157	703	238	61%	21%
Lotus eSuite Chart	1,387	-	633		46%
swingall	1,657	998	338	60%	20%
Mockingbird	2,350	-	506		22%
Reservation System	3,067	-	736		24%
JDK 1.2 runtime	4,652	2,820	1,069	61%	23%

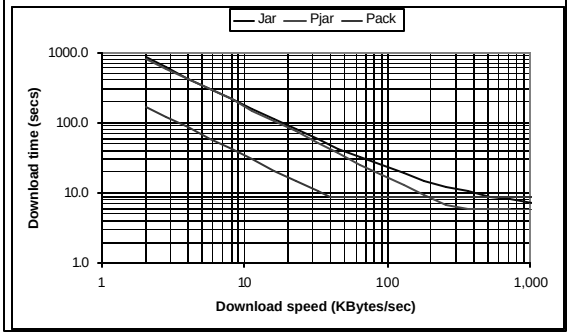
Compression ratios



Execution Times

- For swingall.jar
  - Jar format: 1,657 KBytes
  - compressed size: 338 Kbytes
  - decompression time (Ultra 5 333Mhz)
    - to memory: 3 secs
    - to jar file: ~ 14 secs
  - time to load classes : 5.8 secs
    - time to define, resolve and verify

Download times



Decoder size & security

- Decoder is about 35Kbytes
  - Could be downloaded
    - not useful for small archives
  - Could be installed as extension
- Decoder either needs permission to write to a temporary file or permission to create a class loader
  - Can do this under 1.2 security model

Providing Jar functionality

- Jar archives contain more than class files
  - images, text files, resources
  - manifest (signatures, …)
- Add a stream of non-class files
  - a zip archive, without individual compression but with overall compression

Complication for signatures

- Compression and decompression changes a class file
  - by renumbering the constant pool
- Signatures from source class files won't work on decoded class files
- Decompress once, sign decompressed class files, use those signatures
  - decompression is deterministic

Related work - lots!

- Used few ideas that hadn't been considered previously
- Compression of executable code
  - Ernst, Evans, Fraser, Lucco and Proebsting, PLDI97
- Compression of Java Classfiles
  - Nigel Horspool et al.

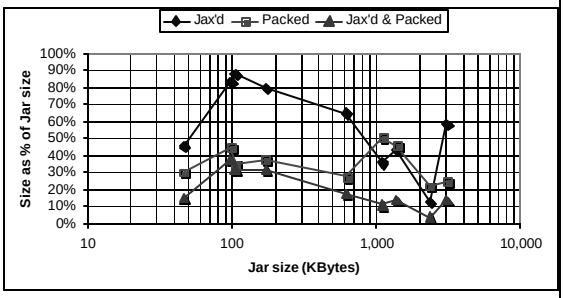
Jax from IBM

- Java Application eXtraction
  - available from www.alphaworks.ibm.com
- Extracts just the classes and methods needed by application
- Very useful if your application uses small part of a large library
  - eliminates need to ship entire library

Combining Jax and Pack

Benchmark	jar size (Kbytes)	Jax'd	Packed	Jax'd & Packed
Hanoi	46	46%	30%	15%
IBM Host on demand	98	84%	45%	37%
ICE Browser	105	87%	35%	32%
JavaFig	171	79%	37%	31%
Cinderella	625	65%	27%	17%
Lotus eSuite Sheet	1,101	35%	50%	11%
Lotus eSuite Chart	1,387	43%	46%	14%
Mockingbird	2,350	13%	22%	4%
Reservation System	3,067	58%	24%	14%

Combining Jax with Packing



Software release

- Codec will be open source
  - want to avoid forking of source
- Alpha release available momentarily
  - Almost certainly has bugs
  - Current format not supported in the future
    - any small tweak changes the format

Getting it ready for the mass market

- Additional work needs to be done
  - Testing
  - User interface
  - Installation/Code signing
- I don't have time to provide customer support
- Looking for partners

Future work

- Compact object serialization formats
- Progressive class file loading
  - Ordering class files
  - Reducing class files loaded but not used
    - some class files loaded only for verification
  - Eagerly load class files when no other work
  - Separating application into modules
    - don't download modules unless needed

Questions?

Slides, software available from:
http://www.cs.umd.edu/~pugh/java