Alibaba Cloud
Worldwide Cloud Services Partner

Optimizing Python startup time by code data share

Yichen Yan

2024.9.18

Alibaba Cloud
Worldwide Cloud Services Partner

- I work at Alibaba Cloud as a senior engineer

- We are the programming languages and compiler team at Alibaba Cloud
- Focusing on languages (C&C++, Java, etc.), runtime (PyTorch, etc.), supporting upper layer cloud and AI infrastructure

- Current products include AJDK (AlibabaJDK), Alibaba GCC/LLVM, AI software stack

code-data-share-for-python

Alibaba Cloud
Worldwide Cloud Services Partner

- https://github.com/alibaba/code-data-share-for-python

1 Current status of Python startup

Status of Python startup

Overview

- Python is popular
- Widely used in startup-time sensitive scenario (e.g. FaaS)

- Recently, Python has been used in complex AI application
- involve a lot of third party packages
- importing `torch` needs ~1s

```
$ time python -c 'import torch'

real    0m1.302s
user    0m2.227s
sys     0m4.783s
```

Status of Python startup

Alibaba Cloud
Worldwide Cloud Services Partner

Abstract import mechanism

- Importing a python package contains:

- If pyc file does not exist, read py file, compile python source code to bytecode, then marshall bytecode to pyc file;
- If pyc file exists, read pyc file, and get bytecode by unmarshalling;
  - pyc file does not store the exact content of bytecode, it's a serialized format;
  - pyc file profides significant speedup (>50%) when launching python application

- After get bytecode, python interpreter shall execute the bytecode to get an in-memory package.

Status of Python startup

Alibaba Cloud
Worldwide Cloud Services Partner

Analysis of `torch` package

- `python –Ximporttime –c ‘import torch’` to see detailed imported package and time
- Python audit event to see file open

- More than 1000 imported (sub)packages
- ~1000 file read
- Presumably more IO op, e.g. stat

```
$ python -Ximporttime -c 'import torch' 2>&1 | wc
   1159    8115   84293
$ cat t.py
import sys
def h(e, args):
    if e != 'open': return
    path, mode, flags = args
    print(path)

sys.addaudithook(h)
import torch
$ python t.py | wc
    960     960   92734
```

Status of Python startup

Analysis of `torch` package (2)

- Importing a package contains three major part

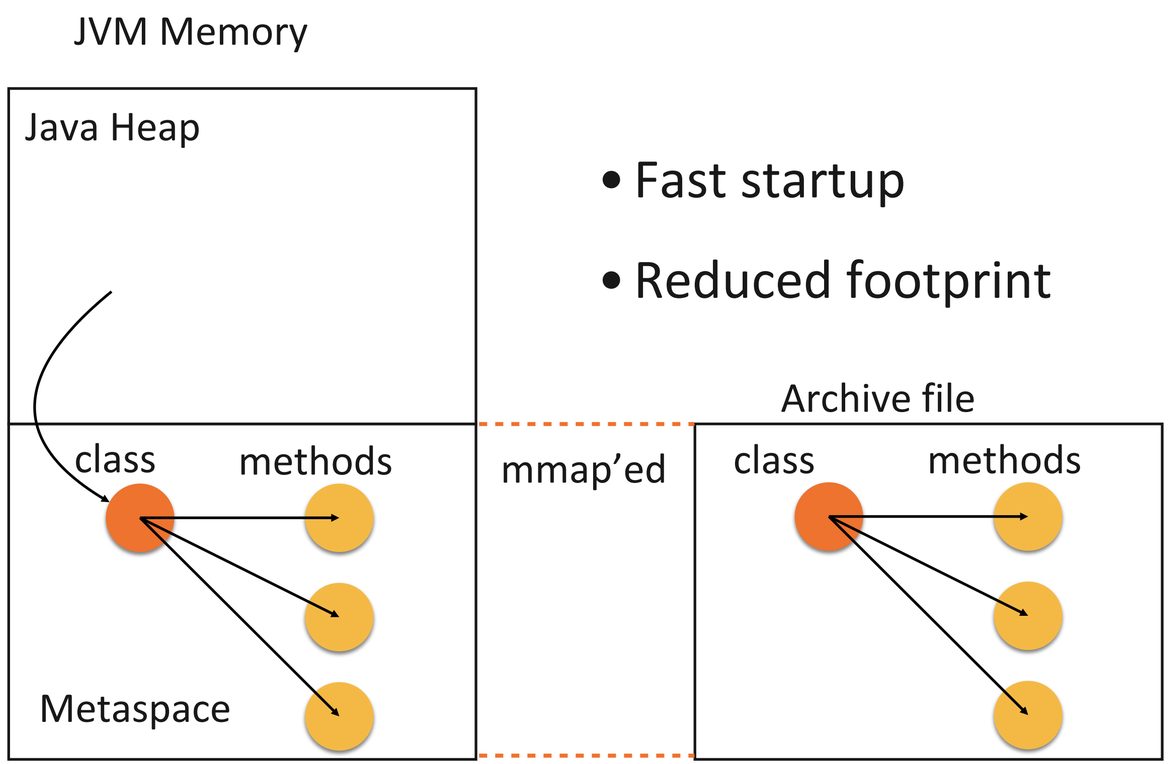
- IO, e.g. read file, check file exists, …
- Unmarshall
- Bytecode execution

- Due to the dynamic feature of Python, bytecode execution is hard to eliminate.
- We focus on trying to reduce the first two part

2 Introducing class data sharing

AppCDS in Java

Location independent class object



- Fast startup
- Reduced footprint

- CDS enabled: pointed to file-mapped memory
- CDS disabled: parsed from class files

History

- Commercial feature in Oracle JDK 8/9
- Open source since OpenJDK 10 (JEP 310, 2018)
- Default CDS Archives since OpenJDK 12 (JEP 341, 2019)

Adoption

- Alibaba Cloud SAE using AppCDS in production env
  - 30% startup time reduction

CDS in Python?

- Python's package is constructed dynamically at runtime, via bytecode
- There's no simple solution to "cache" the actual package

- Reduce overhead to get bytecode

- Memory mapped bytecode
- Single file IO
- No unmarshalling

Module construction in Python

- Code object (`PyCodeObject`)
- `eval_module(code, {})`

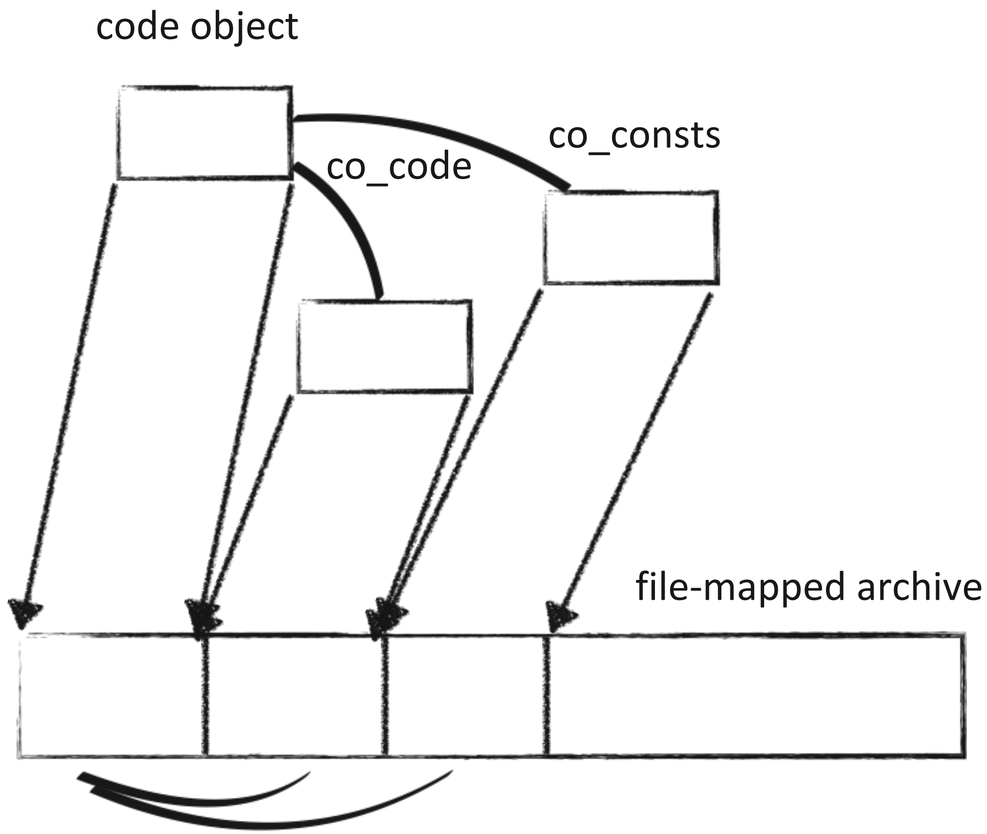
- Python object:
  `string,tuple,bytes,int,float,double,complex,constants`
- C struct: `PyCodeObject`

- Make these data types are memory-mappable

3 Implementation

Memory map

- Copy code object to a mapped memory
- Archive has fixed address

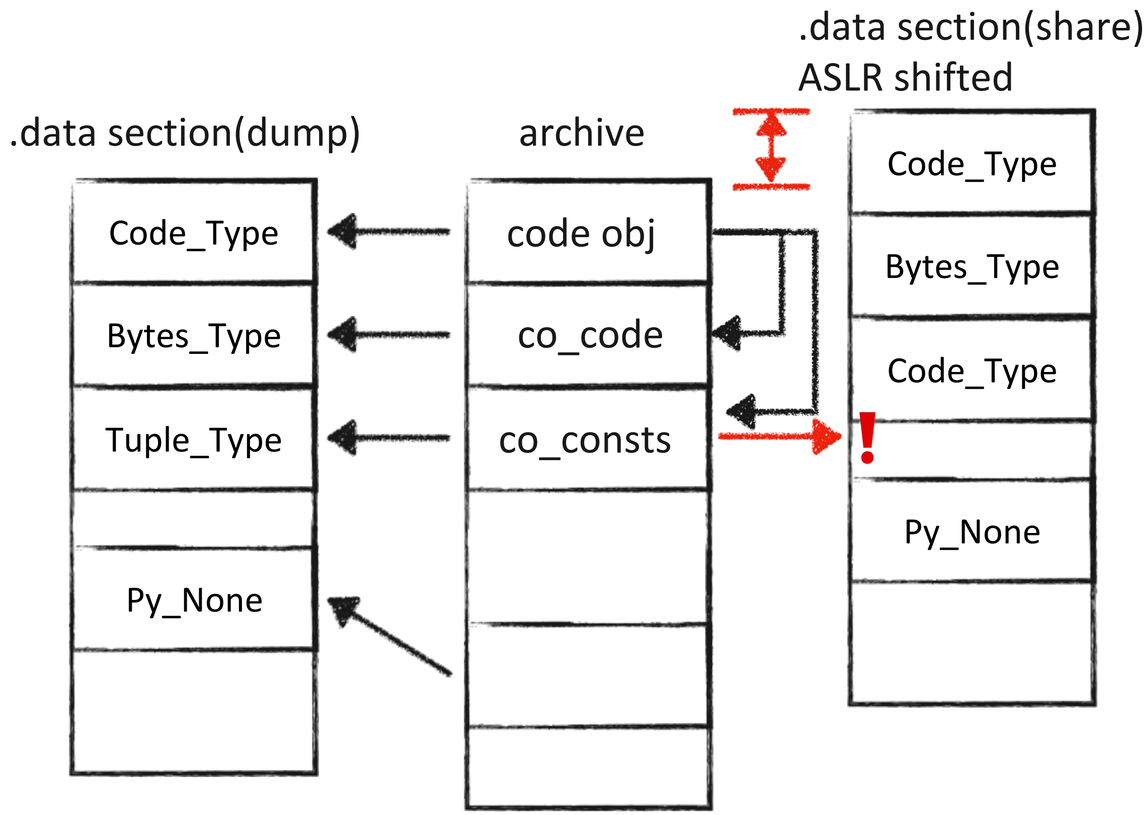


Make a compact memory image by recursive copy.

Memory map (2)

- Address space layout randomization
- ASLR is enabled on most platform

- Patch to-heap reference during loading
- `PyObject.ob_type`
- `PyTypeObject` (ASLR shifted)



MAP_FIXED ensures that the pointer inside the archive is correct, but ASLR causes the pointer to the data section invalid.
Iterating over the object graph to fix the pointers.

GC

- Python uses a reference-based GC

- 3 generations
- Each generation is a linked list, containing all objects in this generation

- Reference counter are managed by interpreter
- When reaching zero, object will be deallocated

- We already these objects are all accessible, and will live “forever”
- Extra reference count for in-archive object to avoid GCed
- Since 3.12, `_Py_SetImmortal`

Implementing each type

- Constants
- `None`
- `True/False`
- `Ellipsis`

```
if (ty == &PyBool_Type || ty == &_PyNone_Type || ty == &PyEllipsis_Type) {
    *target = op;
}
```

Implementing each type

- Constants
- Simple objects (trivial layout)
- `float`
- `complex`

```
#define SIMPLE_MOVE_IN(obj_type, type, copy)                \
    obj_type *res = PyCDS_Malloc(_PyObject_SIZE(type)); \
    PyObject_INIT(res, type);                               \
    do {                                                    \
        copy                                                \
    } while (0);                                            \
    *target = (PyObject *)res;                              \
    UNTRACK(*target);
```

```
else if (ty == &PyComplex_Type) {
    SIMPLE_MOVE_IN(PyComplexObject, &PyComplex_Type,
                   { res->cval = ((PyComplexObject *)op)->cval; })
}
else if (ty == &PyFloat_Type) {
    SIMPLE_MOVE_IN(PyFloatObject, &PyFloat_Type,
                   { res->ob_fval = ((PyFloatObject *)op)->ob_fval; })
}
```

Implementing each type

- Constants
- Simple objects (trivial layout)
- Variable-length objects
- `int`
- `bytes`
- `string(unicode)`

```
    else if (ty == &PyLong_Type) {
        // _PyLong_Copy starts
        PyLongObject *src = (PyLongObject *)op;

#if PY_MINOR_VERSION >= 12…
#endif

        Py_ssize_t size;
        // _PyLong_New starts
#if PY_MINOR_VERSION >= 12…
#else
        size = Py_SIZE(src);
        if (size < 0)
            size = -(size);
        PyLongObject *res = PyCDS_Malloc(offsetof(PyLongObject, ob_digit) +
                                         size * sizeof(digit));
        PyObject_INIT_VAR((PyVarObject *)res, &PyLong_Type, Py_SIZE(src));
        // _PyLong_New ends

        while (--size >= 0) {
            res->ob_digit[size] = src->ob_digit[size];
        }
#endif
        // _PyLong_Copy ends

        *target = (PyObject *)res;
        UNTRACK(*target);
```

Implementing each type

- Constants
- Simple objects (trivial layout)
- Variable-length objects
- Nested object
- `PyTuple_Type`
- `PyFrozenSet_Type`

- Similar with varlen objects
- Allocate n pointer slot first, then copy each object, heap growth after the container object

Implementing each type

- Constants
- Simple objects (trivial layout)
- Variable-length objects
- Nested object
- `PyCodeObject`

```
#define SIMPLE_HANDLER(field)    \
    do {                         \
        res->field = src->field; \
    } while (0)
#define PATCH_HANDLER(field) PYCDS_MOVEIN_REC_RETURN(src->field, &res->field)
        UNWIND_CODE_FIELDS
#undef P
#undef S

#if PY_M


#endif
```

```
#define UNWIND_CODE_FIELDS PATCH_HANDLER(co_consts); PATCH_HANDLER(co_names); IF_11_OR_LATER(PATCH_HANDLER(co_exceptiontable);)
SIMPLE_HANDLER(co_flags); SIMPLE_HANDLER(co_argcount); SIMPLE_HANDLER(co_posonlyargcount); SIMPLE_HANDLER(co_kwonlyargcount);
SIMPLE_HANDLER(co_stacksize); SIMPLE_HANDLER(co_firstlineno); IF_11_OR_LATER(SIMPLE_HANDLER(co_nlocalsplus);)
IF_12_OR_LATER(SIMPLE_HANDLER(co_framesize);) SIMPLE_HANDLER(co_nlocals); IF_11_OR_LATER(SIMPLE_HANDLER(co_ncellvars);)
IF_11_OR_LATER(SIMPLE_HANDLER(co_nfreevars);) IF_12_OR_LATER(NOOP_PLACEHOLDER(co_version);) IF_11_OR_LATER(PATCH_HANDLER(co_localsplusnames);)
IF_11_OR_LATER(PATCH_HANDLER(co_localspluskinds);) PATCH_HANDLER(co_filename); PATCH_HANDLER(co_name);
IF_11_OR_LATER(PATCH_HANDLER(co_qualname);) IF_10_OR_LATER(PATCH_HANDLER(co_linetable);) NOOP_PLACEHOLDER(co_weakreflist);
IF_12_OR_LATER(NOOP_PLACEHOLDER(_co_cached);) IF_12_OR_LATER(NOOP_PLACEHOLDER(_co_instrumentation_version);)
IF_12_OR_LATER(NOOP_PLACEHOLDER(_co_monitoring);) IF_12_OR_LATER(SIMPLE_HANDLER(_co_firsttraceable);) NOOP_PLACEHOLDER(co_extra); IF_11_OR
```

Implementing each type

- Constants
- Simple objects (trivial layout)
- Variable-length objects
- Nested object
- `PyCodeObject`

- rapidly evolving
- New JIT feature

```c
#if PY_MINOR_VERSION >= 11
static inline void
COPY_AND_DEOPT_CODE(PyCodeObject *res, PyCodeObject *src,
                    const Py_ssize_t code_count, const Py_ssize_t code_size)
{
    _Py_CODEUNIT *instructions = _PyCode_CODE(res);
    memcpy(instructions, src->co_code_adaptive, code_size);
    memcpy(instructions, _PyCode_CODE(src), code_size);
```

Patching type pointer

- Calculate shift based on constant address
- Recursively find all type pointer

- Update pointer

```
PyCDS_PatchPyObject(PyObject **ref)
        PyCDS_Verbose(2, "patching basic types.");
        *ref = UNSHIFT(op, cds_status.shift, PyObject);
    }
#if PY_MINOR_VE
    else if (!P
        PyCDS_V
        *ref = UNSHIFT(op, cds_status.shift, PyObject);
        assert(_PyCDS_InPySingleton(*ref) || _PyCDS_MayBeDeepFreeze(*ref));
    }
```

```
#define UNSHIFT(p,shift,ty) ((ty *)(P(p) + (shift)))

Expands to:

((PyObject *)(((p_type)(op)) + (cds_status.shift)))
```

4 Optimization

String intern

- String comparing falls into slow path

- Python holds a dict for all interned string (string singleton)
- Newly create string with same content will be from the dict

- Make string interned as much as possible

- When loading cds archive

1. If a string not interned yet, intern string
2. If the string is already interned, update in-archive reference

- Avoid duplicated string with same content

Page fault

- Whole archive needs to be loaded

- Trigger page fault before really used
- `MAP_POPULATE`
- Manually (write to each byte: `((char volatile *)shm)[i] += 0`)

5 Runtime

Runtime support

- Which packages to archive

- Run real workload first, hook `importlib`, record imported packages,
- Run a special script provided by cds, dump all traced packages into one single archive
- Run with cds, hook `importlib` again, get speedup

Runtime support (FaaS)

- Integrated into FaaS infrastructure
- Users don’t need to manually run application with extra argument
- First run will be traced, archive will be automatically generated
- Further runs will be accelerated

6 Performance

Performance

Benchmark	perf-import-3.11-raw	perf-import-3.11-cds
boto3	121 ms	98.4 ms: 1.23x faster
requests	82.3 ms	67.8 ms: 1.21x faster
numpy	104 ms	84.5 ms: 1.23x faster
pyparsing	52.1 ms	45.8 ms: 1.14x faster
sqlalchemy	164 ms	141 ms: 1.17x faster
werkzeug	76.8 ms	63.9 ms: 1.20x faster
aiohttp	124 ms	105 ms: 1.18x faster
google-cloud-storage	180 ms	151 ms: 1.20x faster
flask	136 ms	112 ms: 1.21x faster
azure-core	81.7 ms	66.9 ms: 1.22x faster
jsonschema	91.5 ms	79.4 ms: 1.15x faster
Pillow	34.0 ms	29.3 ms: 1.16x faster
scipy	112 ms	92.8 ms: 1.20x faster
tensorflow	1.60 sec	1.26 sec: 1.27x faster
seaborn	785 ms	681 ms: 1.15x faster
azureml-core	519 ms	407 ms: 1.28x faster
pytorch	1.03 sec	894 ms: 1.15x faster
Geometric mean	(ref)	1.20x faster

7 Challenges and Future work

Future work

- Python 3.13 support
- How to patch mapped memory faster? static key?
- Benefits from JIT information?
- Archive the real package

Static key: https://docs.kernel.org/staging/static-keys.html


Alibaba Cloud | MORE THAN JUST CLOUD