Actor-d & scala

- 2008/05/10
- scala lift off!
- steve.yen@metaha.com

lessons from a scala server

- infrastructure in scala

about me

- steve yen
- lisp, c/c++/tcl, java, python, ruby/javascript, scala

Actor-d is a...

- memcached server clone
- written in scala
- open source
- actors inside

Actor-d vision

- large mesh of coordinated actors
- distributed, replicated storage
- N >> 1

scala for infrastructure?

- ymmv
- just like java

scala == JPIE ?

- java platform infrastructure edition
- missing...
  - solution to dreaded GC pause
  - actor/process isolation
    - can't just throw these into a yet another scala library, eh??

initial actor-d version

- david pollak's fault
  - “need scala that speaks memcached ABI...”
- ~ 1 weekend+
  - it took longer to pick a name
- built on apache mina NIO
  - via jmemcached

idea: memcached: the next...

- wiki!
- memcached makes an interesting language / framework shootout target?
- pick your weapon of choice...

and write...

- httpd - mid 90's
- wiki - late 90's
- RoR clone - 03's
- memcached clone - ?

memcached properties

- useful, non-toy
- distributed, expirable, share-nothing LRU cache
- simple networking 101
- memslap for perf testing
- many client bindings

how about an erlang version?

- or twisted
- or ocaml
- or STM...
- or scala...
- how close/beyond the original c memcached?

compare apples to oranges

- LoC game time!

original C memcached

- version 1.2.5, wc -l *.c *.h
  - text-based protocol
  - highly tuned, battle tested
  - includes both TCP & UDP transports
- 6311 LoC

jmemcached: java version

- 2109 LoC *.java
  - no binary protocol
  - no UDP protocol, TCP only
- based on apache mina NIO
  - mina related code: ~400+ LoC

actord: scala version

- 3546 LoC *.scala
  - no binary protocol
  - no UDP protocol, TCP only
- includes...
  - range queries
  - cmdline configurable persistence
    - (1423 LoC)
  - and...

actord: menu of networking

- 3 networking implementations
  - apache mina NIO (139 LoC)
  - sun grizzly NIO (196 LoC)
  - blocking socket (155 Loc)
- which is best?
  - blocking sockets “much better” than NIO
  - when # clients < N

so much for “comparing” LoC

onwards to more learnings

- case classes
- functional programming
  - partial functions
- fancy types, generics
- implicits
- scala & JVM
- performance

case classes in scala

- two examples
  - network protocol handling
  - command-line parsing

protocol handling

- memcached & jmemcached...
  - split the incoming line
  - with the first word...
    - if strcmp then
    - else if strcmp then
    - else if strcmp then
    - else if strcmp then
    - else if strcmp then

memcached protocol.txt...

- Retrieval command:
- ------------------
- The retrieval command "get" operates like this:
-
- **get <key>*\r\n**
-
- - <key>* means one or more key strings separated by whitespace.
-
- After this command, the client expects zero or more items, ... After all
- the items have been transmitted, the server sends the string
-
- "END\r\n"

actord protocol parsing

- extensible Spec case class

```
List(
  Spec("get <key>*",
       (svr, cmd) => {
         svr.get(cmd.args).
             foreach(entry => cmd.write(entry, false))
         cmd.reply(END)
       }),
  Spec("delete <key> [<time>] [noreply]",
       (svr, cmd) =>
         cmd.reply(svr.delete(cmd.args(0), cmd.argToLong(1),
cmd.noReply),
                   DELETED, NOT_FOUND)),
```

List(Spec)

- at startup, make a Spec lookup table
  - as messages come in, just lookup Spec's
- keys are Strings
  - “get”, “delete”, “set”
- immutable.Map[String, Spec]
  - import scala.collection._

Whoops, too slow, so...

```
def indexSpecs(specs: List[Spec]): Array[List[Spec]] = {
  // A lookup table by first character of the spec name.
  // Buckets in the lookup table are just Lists.
  val lookup = new Array[List[Spec]](26)
  for (i <- 0 until lookup.length)
    lookup(i) = Nil
  for (spec <- specs) {
    val index = spec.name(0) - 'a'
    lookup(index) = lookup(index) ::: List(spec)
  }
  lookup
}
```

Lookup a Spec

```
def findSpec(x: Array[Byte], xLen: Int,
             lookup: Array[List[Spec]]): Option[Spec] =
  lookup(x(0) – 'a').find(
    spec => arrayCompare(spec.nameBytes,
                         spec.nameBytes.length,
                         x, xLen) == 0)
```

- *Also, I **switched** from String to Array[Byte]*
- *Also, cannot have a Map[Array[Byte], Spec]*

command-line parsing

- memcached cmdline parsing
  - classic getopt
  - while-switch-case loop on first character

jmemcached cmdline parsing

```
import org.apache.commons.cli.*;
Options options = new Options();
options.addOptions("h", "help", false, "print this help screen");
options.addOptions(...);
options.addOptions(...);

CommandLineParse parser = new PosixParser()
CommandLine cmdLine = parser.parse(options, args)
if (cmdLine.hasOption("help")) then
else if (...) then
```

actord cmdline parsing

- wrote a new parser: 53 LoC

```
def flags = List(
  Flag("limitMem", "-m <num>" :: Nil, "Use <num> MB memory max"),
  Flag("port", "-p <num>" :: Nil, "Listen on port <num>"),
  ...
)

val flagValues = parseFlags(args, flags)
```

To access an arg

```
val port = arg("port", "11211").toInt
val limitMem = arg("limitMem", "64").toLong
```

- So, what's parseArgs() look like...

hello, deep indentation

```
def parseFlags(args: Array[String], flags: List[Flag]): List[FlagValue] = {
    val xs = (" " + args.mkString(" ")).split(" -")
    if (xs.headOption.
           map(_.trim.length > 0).
           getOrElse(false))
        List(FlagValue(FLAG_ERR, xs.toList))
    else
        xs.drop(1).
           toList.
           map(arg => {
               val argParts = ("-" + arg).split(" ").toList
               flags.find(_.aliases.contains(argParts(0))).
                     map(flag => if (flag.check(argParts))
                                     FlagValue(flag, argParts.tail)
                                 else
                                     FlagValue(FLAG_ERR, argParts)).
                     getOrElse(FlagValue(FLAG_ERR, argParts))
           })
}
```

the other Specs

- Specs testing framework
  - rewrote all SUnit tests into Specs
  - reads better
  - better mvn support

functional programming

actord memory structures

- scala.collection.
  - immutable.SortedMap[String, MEntry]
- range queries!
- 1st weekend...
  - immutable.TreeMap - red-black tree
- immutability
  - favor many readers
  - one TreeMap per CPU for less hot roots

ooops

- can't persist TreeMap
  - can't subclass/extend internal nodes
  - wasn't designed to swizzle
- answer: clone code
  - what are these? <% <: <+
    - dangerous looking ascii art
  - covariant, contravariant?
  - -S, +T ?

Treap.scala

- randomized binary balanced
- tree + heap
  - elegant, original from ML
- leverage heap priority to bubble hot data to root
  - or, to encode expiry

persistable StorageTreap

- simple design
- log structured
  - append only files
  - less corruptible
- rotatable
- swizzle nodes in & out

scala & JVM

- JVM tools work fine
- profiling / heap monitors
- tools that rewrite bytecode
- agentlib tools work fine
  - hprof
  - jrat

* aside: hibernate & scala

- hibernate rewrites bytecode
- hibernate expects certain pojo naming conventions
- scala val and var fields not what default hibernate wants
- **use @BeanProperty annotation!**

scala has JVM's issues

- warm up time (hotspot)
- GC pauses
- GC memory hunger
  - 5x memory vs c??

performance vs c memcached

- suspect #'s
  - single client
  - multiple clients
- actord has higher variance
  - GC suspected
  - cacheline misses suspected

leverage JVM knowledge

- String and obj creation avoidance in main loops
  - CharSet decode/encode
  - message.split(" ")
  - use Array[Byte] instead
- GC is good, mostly
  - except for core main loop!
- can't extend String / Array

mmap still broken

- not a scala problem

plus, new scala'isms

- after performance profiling, look for...
  - auto-boxing / implicit issues
  - RichFoo proliferation
  - RichString's!
    - key1.compare(key2)

answer: OString as keys

- from SortedMap[String, MEntry]
- to SortedMap[OString, MEntry]
- but most of the codebase still passes around Strings, not OStrings

- implicits to the rescue
- cost: many OString wrappers
  - that might mature into permanent gen
  - instead of many many many many transient RichString wrappers

implicit to the rescue

- for this change, implicit and static typing was a huge win
- but how about implicit for for large projects?

scala collection iterators

- Seq.indexOf
- someBytes.indexOf(' ')
  - it's doing Iterator / hasNext underneath the hood, rather than an Array index

actors in actord

- one persister actor spills data to disk asynchronously
- another compactor actor vacuums files asynchronously

And, an actor per cpu...

- SortedMap per CPU
- controls all SortedMap modifications
- send async/sync messages
- writes are serialized
- readers have MVCC and hardly block

But

- still had to remove actors from main GET message loop
- want:
  - asynchronous stats and LRU updates
  - asyncActor ! UpdateLRUAndStats(...)
- too slow - 30% hit

Instead

- replaced with synchronized
  - faster
  - but still need to verify multi-cpu core scalability?
  - will try java.util.concurrent later

`receive` wins over `react`

- Much faster

Partial Functions

- for actord extensibility
  - based on lift design pattern
- can use actord as just a jar file and still change behavior
- removed PF's too in main loop
  - now, just use subclassing and method overriding for extensibility

the horse's teeth

- what's the performance sniff test for a project?
  - answer: look at its util dir/file

actord util code

- mostly Array[Byte] manipulation
  - **arraySplit**(a: Array[Byte], offset: Int, len: Int, x: Byte)
  - **arrayIndexOf**(a: Array[Byte], offset: Int, length: Int, x: Byte): Int
  - **arrayCompare**(a: Array[Byte], aLength: Int, b: Array[Byte], bLength: Int): Int

break

- instead of break statement
- just create a helper func with the loop
- and use **return**
- closures avoid info passing
- example...

Conclusion

- Love Scala
  - actors, RichStrings, implicits, autoboxing, partial functions, closures/anon-funcs, Option, DSL's
  - I used all these new features and more
- Must profile, and then...
  - Your server's main core loop might devolve back into just slinging around Array[Byte]'s

That's it! Thanks!

- code.google.com/p/actord

extra...

actord: feature equivalent to jmemcached

- remove range queries
- remove persistence
- just blocking socket I/O
  - ~1790 LoC *.scala

memcachedb - memcached + persistence

- 4065 LoC
- built on berkeley db

misc

- maven is your friend
- except it's so verbose and undocumented