Using D for Development of Large Scale Primary Storage

Liran Zvibel
Weka.IO, CTO
liran@weka.io
@liranzvibel

#DConf2016

Agenda

- Weka.IO Introduction
- Our progress since we picked off
- Examples where D really shines
- Our challenges
- Improvements suggestions
- Q&A

Weka.IO Introduction

About Weka.IO

- Enabling clouds and enterprises with a single storage solution for resilience, performance, scalability and cost efficiency
- HQ in San Jose, CA; R&D in Tel Aviv, Israel
- 30 engineers, vast storage experience
- VC backed company; Series B led by Walden International; Series A led by Norwest Venture Partners
- Product used in production by early adopters (still in stealth)
- Over 200k loc of our own D code, about 35 packages

Storage system requirements

- Extremely reliable, “always on”, state-full.
- High performance data path, measured in µsecs
- Complicated “control path”/“management code”
- Distributed nature due to HA requirements
- Low level interaction with HW devices
- Some kernel-level code, some assembly
- Language has to be efficient to program, and fit for large projects

The Weka.IO framework

- Software only solution
- User-space processes
- 100% CPU, polling based on networking and storage
- Asynchronous programming model, using Fibers and a Reactor
- Memory efficient, zero-copy everything, very low latency
- GC free, lock-free efficient data structures
- Proprietary networking stack from Ethernet to RPC

Our Progress

Current state for Weka

- No more show-stoppers, still a long way to go
- Indeed productivity is very high, very good code-to-features ratio
- We are able to “rapid prototype” features and then iron them
- All major runtime issues resolved
- We get great performance

- Choosing D was a good move, and proved to be a huge success

Compilation progress

- Switched to LDC (thanks David Nadlinger and the LDC team!)
- Compilation is now by package
  - Better RAM “management”
  - Leveraging parallelism to speed build time
- Recent front-ends “feel” much more stable
- LDC lets us build optimized compilation with asserts, which is a good thing for QA.

LDC status

- Got over 100% performance boost over DMD
  - When compiling as a single package with optimizations
- Fiber switching based on registers and not pthreads
- No GC allocation when throwing and handling exceptions (Thanks Mithun!)
- Integrate `libunwind` with dwarf support for stack traces (no `--disable-fp-elim`)
- Support debug (`-g`) with backend optimizations
- Template instantiation bug — still unresolved for the upstream
- `@ldc.attribute.section("SECTIONNAME")`
- `-static` flag to ldc, allowing easy compile and shipment of utilities

GC allocation and latency

- We now check how much we allocated (using hacks, api would be nice) from the Reactor, and decide to collect if we allocated more than 20MB
- Collection actually happens very infrequently (few times in an hour)
- Collection time is de-synchronized across the cluster
- Collection time still significant — about 10ms
- Main drawback — allocation MAY take ‘infinite’ amount of time if kernel is stressed on memory.

Exceptions and GC

- Exception handling code was modified to never rely on GC allocation
- Reactor and Fibers code (+ our `TraceInfo` class) modified to keep the trace in a fiber local state.
  - ✳ Problem: potentially throwing from `scope(exit/success/failure)`
- `Throwable`s are a class, so allocating them comes from the GC, must be statically allocated:
  - `static __gshared auto ex = new Exception(“:o(”);`

Code Tidbits

NetworkBufferPtr

```
@nogc @property inout(NetworkBuffer)* get() inout nothrow pure {
    auto ptr = cast(NetworkBuffer*)(_addr >> MAGIC_BITS);
    assert (ptr is null || (_addr & MAGIC_MASK) == ptr._gen);
    return ptr;
}

alias get this;
```

- _gen keeps incrementing when buffets allocated from pools
- Pointers remember their generations, and validate accurate access
- Helps debugging stale pointers
- problem with implicit casts of `null`, alias this is not strong enough. Maybe some syntax could help

Handling all enum values

```
switch (pkt.header.type) {
    foreach(name; __traits(allMembers, PacketType)) {
        case __traits(getMember, PacketType, name):
            return __traits(getMember, this, "handle" ~ name)(pkt);
}
```

- Similar solution verifies all fields in a C struct have the same offset, naturally the C part ends up being much more complex.

Flag setting/testing

```
@property bool flag(string NAME)() {
    return (_flags & __traits(getMember, NBFlags, NAME)) != 0;
}
@property void flag(string NAME)(bool val) {
    if (val) {
        _flags |= __traits(getMember, NBFlags, NAME);
    } else {
        _flags &= ~__traits(getMember, NBFlags, NAME);
    }
}

buffer.flag!"TX_ACK" = true;
```

Efficient packing

```
static if (JoinedKV.sizeof <= CACHE_LINE_SIZE) {
    alias KV = JoinedKV;
    enum separateKV = false;
} else {
    struct KV {
        K key;
        /* values will be stored separately for
           better cache behavior */
    }
    V[NumEntries] values;
    enum separateKV = true;
}
```

Challenges

Compilation time

- Project is broken into ~35 packages.
- Some logical packages are compiled as several smaller packages
- Current 2.0.68.2 compiler has several packages compiled about 90 seconds, leading to total compile time of 4-5 minutes.
- Newer 2.070.2+PGO compiler reduces time by about 35% (Thanks Johan!) . Still getting 3-4 minutes per complete compile.

Compile time improvement suggestions

- Introduce more parallelism into the build process
- Support incremental compiles.
  - Now when a dependency is changed, complete packages have to be completely rebuilt. In many cases, most of the work is redundant
  - When dependency IMPLEMENTATION is changed, still everything gets recompiled
- Support (centralized) caching for build results.

- Don't let humans “context switch” while waiting for the compiler!

Long Symbols

- Total symbols: 99649, over 1k: 9639, over 500k: 102, over 1M: 62
- Longest symbol was 5M!

- Makes working with standard tools much harder (some nm tools crash on the exe).
- A simple hashing solution was implemented in our special compiler
- Demangling now stopped working for us, we only get module/func name

- More time is spent on hashing than what is saved on linkage. We may need a "native" solution.

Phobos Algs Forcing GC

```
private struct MapResult(alias fun, Range, ARGS…) {
    ARGS _args;
    alias R = Unqual!Range;
    R _input;
    this(R input, ARGS args) {
        _input = input;
        _args = args; }
    @property auto ref front(){ return fun(_input.front, _args); }
…
auto under_value_gc(R)(R r, int value) { return r.filter!(x => x < value); }

auto under_value_nogc(R)(R r, int value) {return r.xfilter!((x,y) => x < y)(value);}

auto multiple_by_gc(R)(R r, int value) { return r.map!(x => x * value); }

auto multiple_by_nogc(R)(R r, int value) { return r.xmap!((x,y) => x * y)(value); }
```

Improvement Ideas

static foreach

- Make it explicit
- Allow it to manipulate types, to replace complex template recursion

```
template hasUDAttributeOfType(T, alias X) {
    alias attrs = TypeTuple!(__traits(getAttributes, X));

    template helper(int i) {
        static if (i >= attrs.length) {
            enum helper = false;
        } else static if (is(attrs[i] == T) || is(typeof(attrs[i]) == T)) {
            static assert (!helper!(i+1), "More than one matching attribute: " ~ attrs.stringof);
            enum helper = true;
        } else {
            enum helper = helper!(i+1);
        }
    }
    enum hasUDAttributeOfType = helper!0;
}
```

Transitive @UDA

- Specify some @UDAs as transitive, so he compiler can help "prove" correctness.
- For example:
  - Define function as `@atomic` if it does not context switch
  - Function may be `@atomic` if it only calls `@atomic` functions
  - Next step would be to prove that no context switch happens
- Can be implemented in "runtime" if there is a `__traits` that returns all the functions that a function may call.
- Next phase would be to be able to 'prove' things on the functions, so `@nogc`, `nothrow`, `pure` etc can use the same mechanism.

Other Suggestions

- `__traits` that returns that max stack size of a function
- Add a predicate that tells whether there is an existing exception currently handled

- Donate Weka's @nogc 'standard library' to Phobos:
  - Our Fiber additions into Phobos (throwInFiber, TraceInfo support, etc) [other lib funs as well]
  - Containers, algorithms, lockless data structures, etc…

Peta

Exa

Zetta

Yotta

Xenna

Weka (10^{30})

Questions?

WEKA.IO

Table 1

0.68	0.70	0.70 + PGO
88.4	58.1	54.7
84.3	57.1	54.7
75.8	51.0	49.7
67.5	40.7	37.3
59.5	36.4	43.1
56.6	38.3	35.3
51.9	35.9	32.4
50.4	30.9	34.6
44.9	25.2	26.8
42.7	31.0	27.0
35.7	31.3	30.2
35.1	24.5	22.9
31.4	21.1	17.7
30.5	20.5	19.9
25.8	20.1	16.3
19.0	13.5	15.4
18.3	12.0	11.3
14.3	10.6	10.4
13.7	14.0	13.6
9.4	6.8	6.3

- 2.0.70.2 is a major improvement in compile time over the 2.068.2
- Still, the 30-40% improvement mean that engineers have to wait long minutes to get the whole exe to build.

- We're breaking large package into smaller ones, when possible