Kirit Sælensminde

Designing APIs For Performance

2 things to think about

2 Things

Design time

2 Things

Design time

They’re not rules, they’re not principles

2 Things

Design time

They’re not rules, they’re not principles

Not the only things to think about

2 Things

Design time

They’re not rules, they’re not principles

Not the only things to think about

Are really one thing

2 Things

Design time

They’re not rules, they’re not principles

Not the only things to think about

Are really one thing

They are not new things

Motivation: Fostgres

```
CREATE TABLE tags (
 film text NOT NULL REFERENCES films,
 tag text NOT NULL,
 CONSTRAINT film_tags_pk
  PRIMARY KEY (film, tag)
);
```

Motivation: Fostgres

```
{
  "path": [1, "/tags"],
  "GET": "SELECT tag FROM tags WHERE film=$1",
  "PUT": {
    "table": "tags",
    "existing": "SELECT tag FROM tags
            WHERE film=$1",
    "delete": “DELETE FROM tags
            WHERE film=$1 AND tag=$2",
    "columns": {
      "film": {"key": true, "source": 1},
      "tag": {"key": true}
    }
  },
  "DELETE": "DELETE FROM tags WHERE film=$1"
}
```

```
CREATE TABLE tags (
 film text NOT NULL REFERENCES films,
 tag text NOT NULL,
 CONSTRAINT film_tags_pk PRIMARY KEY
  (film_slug, slug)
);
```

Motivation: Fostgres

```
{
  "path": [1, "/tags"],
  "GET": "SELECT tag FROM tags WHERE film=$1",
  "PUT": {
    "table": "tags",
    "existing": "SELECT tag FROM tags
            WHERE film=$1",
    "delete": “DELETE FROM tags
            WHERE film=$1 AND tag=$2",
    "columns": {
      "film": {"key": true, "source": 1},
      "tag": {"key": true}
    }
  },
  "DELETE": "DELETE FROM tags WHERE film=$1"
}
```

```
CREATE TABLE tags (
 film text NOT NULL REFERENCES films,
 tag text NOT NULL,
 CONSTRAINT film_tags_pk PRIMARY KEY
  (film_slug, slug)
);
```

GET /terminator/tags

```
"tag"
"action"
"adventure"
"sci-fi"
"robots"
```

CSV & CSJ

```
foo,bar
one,two
1,2.5
,
t,f
```

CSV & CSJ

```
foo,bar
one,two
1,2.5
,
t,f
```

```
“foo”,”bar”
”one”,”two”
1,2.5
null,””
true,false
```

Let's start

```
for ( auto &row : rs ) {
    bool first{true};
    for ( const json &field : row ) {
        if ( not first ) {
            std::cout << ',';
        } else {
            first = false;
        }
        std::cout << stringify(field);
    }
    std::cout << std::endl;
}
```

Let’s start

```
$ ./real/select.endl "select * from films limit 3"
"id","adult","name_en","popularity","blob"
601,false,"E.T. the Extra-Terrestrial",13.027000,
{"adult":false,"id":601,"original_title":"E.T. the
Extra-Terrestrial","popularity":13.027000,"video":false}
602,false,"Independence Day",14.504000,
{"adult":false,"id":602,"original_title":"Independence
Day","popularity":14.504000,"video":false}
603,false,"The Matrix",23.470000,
{"adult":false,"id":603,"original_title":"The
Matrix","popularity":23.470000,"video":false}
```

Let's start

```
$ ./real/select.endl "select count(*) from films"
"count"
402343
$ time ./real/select.endl "select * from films" >
/dev/null

real    0m7.507s
user    0m7.027s
sys     0m0.261s
```

JSON

```
class json {




  using value_type = std::variant<
    std::monostate,

                                  >};
```

JSON

```
class json {




  using value_type = std::variant<
    std::monostate,
    bool

                                >};
```

JSON

```
class json {

  using value_type = std::variant<
    std::monostate,
    bool, int64_t, double,
                                >};
```

JSON

```
class json {

  using value_type = std::variant<
    std::monostate,
    bool, int64_t, double,
    std::string,
                              >};
```

JSON

```
class json {
  using array_type =
    std::vector<std::shared_ptr<json>>;



  using value_type = std::variant<
    std::monostate,
    bool, int64_t, double,
    std::string,
    array_type                >};
```

JSON

```
class json {
  using array_type =
    std::vector<std::shared_ptr<json>>;
  using object_type =
    std::unordered_map<
      std::string, std::shared_ptr<json>>;
  using value_type = std::variant<
    std::monostate,
    bool, int64_t, double,
    std::string,
    array_type, object_type>};
```

JSON

```
class json {
  using array_type =
    std::shared_ptr<std::vector<json>>;
  using object_type =
    std::shared_ptr<std::map<
      std::string, json>>;
  using value_type = std::variant<
    std::monostate,
    bool, int64_t, double,
    std::shared_ptr<std::string>,
    array_type, object_type>};
```

Remove std::endl

```
for ( auto &row : rs ) {
    bool first{true};
    for ( const json &field : row ) {
        if ( not first ) {
            std::cout << ',';
        } else {
            first = false;
        }
        std::cout << stringify(field);
    }
    std::cout << '\n';
}
```

Remove std::endl

```
$ time ./real/select.endl "select * from films" > /dev/null

real    0m7.568s
user    0m6.981s
sys     0m0.319s
$ time ./real/select.simple "select * from films" > /dev/null

real    0m7.154s
user    0m6.868s
sys     0m0.048s
```

Allocating Memory

```
const auto trial = [](const std::size_t size) {
    for ( auto left = 10 << 20 /* 10MB */; left; left -= size ) {
        if ( not malloc(size) ) std::exit(2);
    }
};
```

Allocation size	Allocations	Total time	Time per alloc	Time per byte
1KB	10240	5401µs	527ns	515ps
16KB	640	1338µs	2090ns	127ps
512KB	20	69µs	3173ns	6ps

The two things: The First

Allocate memory as infrequently as possible

Performing IO

```
std::vector<char> data(10 << 20); // 10MB
std::ifstream("/dev/urandom", std::ios::binary).read(data.data(), data.size());
const auto trial = [&data](std::size_t size) {
    std::ofstream out("random.dat", std::ios::binary);
    for ( std::size_t i{}; i < data.size(); i += size ) {
        out.write(&data[i], size);
    }
};
```

Write size	Writes	Total time	Time per write	Time per byte
1KB	10240	21ms	2µs	2080ps
16KB	640	9ms	15µs	932ps
512KB	20	9ms	480µs	916ps

The two things: The Second

Perform IO as infrequently as possible

The two things

Allocate memory as infrequently as possible

Perform IO as infrequently as possible

Remove std::endl

```
for ( const auto &row : rs ) {
    bool first{true};
    for ( const json &field : row ) {
        if ( not first ) std::cout << ',';
        else first = false;
        std::cout << stringify(field);
    }
    std::cout <<  '\n';



}
std::cout << buffer;
```

Stringification

```
std::string stringify(const json &v) {
    struct stringer {
        std::string operator () (bool b) { return b ? "true" : "false”; }
        std::string operator () (double d) { return std::to_string(d); }
        std::string operator () (const json::array_type &a) {
            std::string s;
            for ( const auto &i : a ) {
                if ( not s.empty() ) s += ',';
                s += stringify(*i); }
            return "[" + s + "]";
        }
        // …
    };
    return std::visit(stringer{}, v.value);
}
```

Stringification

```
void stringify(const json &v, std::string&s) {
    struct stringer {
        std::string &s;
        void operator () (bool b) { s += b ? "true" : "false"; }
        void operator () (const json::array_type &a) {
            s += '['; bool first{false};
            for ( const auto &i : a ) {
                if ( not first ) s += ','; else first = false;
                stringify(*i, s);
            }
            s += ']';
        }
        // …
    };
    std::visit(stringer{s}, v.value);
}
```

Remove std::endl

```
for ( const auto &row : rs ) {
    bool first{true};
    for ( const json &field : row ) {
        if ( not first ) std::cout << ',';
        else first = false;
        std::cout << stringify(field);
    }
    std::cout <<  '\n';


}
std::cout << buffer;
```

Using std::string as a buffer

```
std::string buffer;
for ( const auto &row : rs ) {
    bool first{true};
    for ( const json &field : row ) {
        if ( not first ) buffer += ',';
        else first = false;
        stringify(field, buffer);
    }
    buffer += '\n';
    if ( buffer.size() > (60 << 10) ) {
        std::cout << buffer; buffer.clear(); }
}
std::cout << buffer;
```

Using std::string as a buffer

```
$ time ./real/select.endl "select * from films" > /dev/null

real    0m7.568s
user    0m6.981s
sys     0m0.319s
$ time ./real/select.simple "select * from films" > /dev/null

real    0m7.154s
user    0m6.868s
sys     0m0.048s
$ time ./real/select.buffer "select * from films" > /dev/null

real    0m6.674s
user    0m6.339s
sys     0m0.105s
```

Unbuffered

```
std::setvbuf(stdout, NULL, _IONBF, 0);
std::string buffer;
for ( const auto &row : rs ) {
    bool first{true};
    for ( const json &field : row ) {
        if ( not first ) buffer += ',';
        else first = false;
        stringify(field, buffer);
    }
    buffer += '\n';
    std::cout << buffer;
    buffer.clear();
}
```

Unbuffered

```
$ time ./real/select.simple "select * from films" > /dev/null

real    0m7.154s
user    0m6.868s
sys     0m0.048s
$ time ./real/select.buffer "select * from films" > /dev/null

real    0m6.674s
user    0m6.339s
sys     0m0.105s
$ time ./real/select.unbuffered "select * from films" > /dev/null

real    0m8.763s
user    0m7.421s
sys     0m1.105s
```

Unbuffered

```
# \d pgbench_accounts
             Table "public.pgbench_accounts"
  Column  |     Type      | Collation | Nullable | Default
----------+---------------+-----------+----------+---------
 aid      | integer       |           | not null |
 bid      | integer       |           |          |
 abalance | integer       |           |          |
 filler   | character(84) |           |          |
Indexes:
    "pgbench_accounts_pkey" PRIMARY KEY, btree (aid)
# select count(*) from pgbench_accounts;
  count
---------
 2000000
(1 row)
```

Unbuffered

```
$ time ./real/select.unbuffered "select * from pgbench_accounts" > /dev/null

real    0m14.526s
user    0m9.995s
sys     0m3.907s
$ time ./real/select.simple "select * from pgbench_accounts" > /dev/null

real    0m8.658s
user    0m7.931s
sys     0m0.214s
$ time ./real/select.buffer "select * from pgbench_accounts" > /dev/null

real    0m7.762s
user    0m7.099s
sys     0m0.249s
```

Unbuffered

```
$ time ./real/select.unbuffered "select *, filler, filler from pgbench_accounts" > /dev/
null


real    0m29.667s
user    0m23.299s
sys     0m5.947s
$ time ./real/select.simple "select *, filler, filler from pgbench_accounts" > /dev/null

real    0m22.098s
user    0m21.173s
sys     0m0.506s
$ time ./real/select.buffer "select *, filler, filler from pgbench_accounts" > /dev/null

real    0m19.302s
user    0m18.268s
sys     0m0.493s
```

The Need for Speed

Building a library around 2 things

How Postgres sends data

Byte 0 – Frame type

- “D” for “data row”
- “C” for “command complete”

How Postgres sends data

Byte 0 – Frame type

- “D” for “data row”
- “C” for “command complete”

Bytes 1 to 5 – Frame size

- Frame size in bytes (doesn’t include byte 0)

How Postgres sends data

Byte 0 – Frame type

- “D” for “data row”
- “C” for “command complete”

Bytes 1 to 5 – Frame size

- Frame size in bytes (doesn’t include byte 0)

Bytes 6 to n – Frame data

- Field data Is text

How pgasio handles data

First idea

- 1 read for 5 byte frame header
- 1 allocation + 1 read for body

How pgasio handles data

First idea

- 1 read for 5 byte frame header
- 1 allocation + 1 read for body

Better

- 1 allocation for body + 5 byte header of following frame
- 1 read for body + 5 byte header of following frame

How pgasio handles data

First idea

- 1 read for 5 byte frame header
- 1 allocation + 1 read for body

Better

- 1 allocation for body + 5 byte header of following frame
- 1 read for body + 5 byte header of following frame

Best

Allocate a ton of memory

Read as much data as the kernel has

How pgasio handles data

```
$ time ./real/select.simple "select * from films" > /dev/null

real    0m7.154s
user    0m6.868s
sys     0m0.048s
$ time ./real/select.buffer "select * from films" > /dev/null

real    0m6.674s
user    0m6.339s
sys     0m0.105s
$ time ./real/select.unbuffered "select * from films" > /dev/null

real    0m8.763s
user    0m7.421s
sys     0m1.105s
$ time ./real/select.pgasio "select * from films" > /dev/null

real    0m0.461s
user    0m0.202s
sys     0m0.062s
```

How pgasio handles data

```
$ time ./real/select.unbuffered "select * from pgbench_accounts" > /dev/null

real    0m14.526s
user    0m9.995s
sys     0m3.907s
$ time ./real/select.simple "select * from pgbench_accounts" > /dev/null

real    0m8.658s
user    0m7.931s
sys     0m0.214s
$ time ./real/select.buffer "select * from pgbench_accounts" > /dev/null

real    0m7.762s
user    0m7.099s
sys     0m0.249s
$ time ./real/select.pgasio "select * from pgbench_accounts" > /dev/null

real    0m1.876s
user    0m1.389s
sys     0m0.114s
```

How pgasio handles data

```
$ time ./real/select.unbuffered "select *, filler, filler from pgbench_accounts" > /dev/null

real    0m29.667s
user    0m23.299s
sys     0m5.947s
$ time ./real/select.simple "select *, filler, filler from pgbench_accounts" > /dev/null

real    0m22.098s
user    0m21.173s
sys     0m0.506s
$ time ./real/select.buffer "select *, filler, filler from pgbench_accounts" > /dev/null

real    0m19.302s
user    0m18.268s
sys     0m0.493s
$ time ./real/select.pgasio "select *, filler, filler from pgbench_accounts" > /dev/null

real    0m3.636s
user    0m2.894s
sys     0m0.275s
```

How pgasio handles data

```
struct block {
    std::vector<unsigned char> data;
    std::vector<std::string_view> fields;
};

for (std::size_t row{}; row * cols < b.fields.size(); ++row) {
    for (std::size_t field{}; field < cols; ++field) {
        buffer += process(b.fields[row * cols + field]);
    }
}
```

Share all the things

```
template<typename T>
class vector {
    std::size_t size_;
    T *data_;
};

class string_view {
    std::size_t size_;
    const char *data_;
};
```

Share all the things

```
template<typename T>
class vector {
    std::size_t size_;
    T *data_;
};

class string_view {
    std::size_t size_;
    const char *data_;
};
```

```
template<typename T>
class shared_vector {
    std::size_t size_;
    std::shared_ptr<const T> data_;
};

class shared_string {
    std::size_t size_;
    std::shared_ptr<const char> data_;
};
```

Share all the things

```
shared_vector split(const std::size_t s) {
  auto r = shared_vector{data_, s};
  data_ = std::shared_ptr<const T>{
                    data_, data_.get() + s};
  return r;
}
```

Share all the things

```
shared_vector<shared_string> b = …;
```

Share all the things

```
shared_vector<shared_string> b = …;


for ( auto row{b.split(cols)};
          row.size(); row = b.split(cols) ) {
    for ( auto field : row ) {
        process(field, buffer);
    }
}
```

Share all the things

```
template<typename Allocator>
shared_vector(std::vector<T, Allocator> v)
: size_{v.size()},
    data_{v.data(),
      [o = std::move(v)](const auto &&) {}}
{
}
```

Share all the things

```
$ time ./real/select.unbuffered "select *, filler, filler from pgbench_accounts" > /dev/null

real    0m29.667s
user    0m23.299s
sys     0m5.947s
$ time ./real/select.pgasio "select *, filler, filler from pgbench_accounts" > /dev/null

real    0m3.636s
user    0m2.894s
sys     0m0.275s
$ time ./libs/pgasio/examples/pgasio-csj "select *, filler, filler from pgbench_accounts" > /dev/null
SELECT to CSJ

real    0m2.629s
user    0m3.482s
sys     0m0.669s
```

Designing APIs for performance

War on memory allocations
War on IO

Take them into account when you design

k@kirit.com
Twitter: @KayEss
https://kirit.com
https://github.com/KayEss