

- Node.js-based
- Distributed MySQL Proxy



王达心
@daxin11
2012/09

About Me

- sina, distributed platform dev
- 5 years+ C++ Experiences
- Using node.js no more than 1 year, No experience with JS
- Last company: Baidu, Automatic DevOps Platform, Monitoring Platform



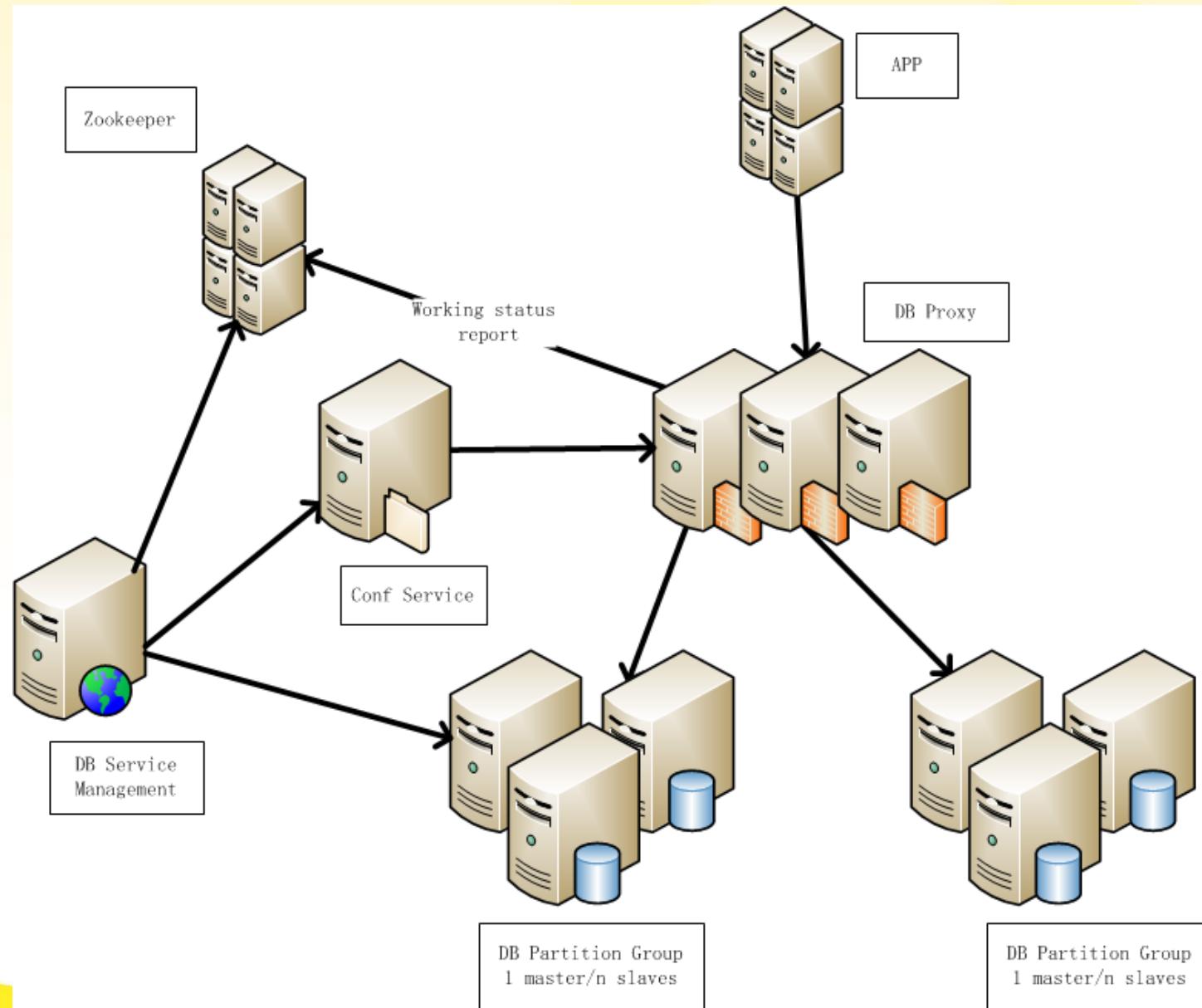
contents

- **What's the problem**
- **Platform Architecture**
- **Proxy functionalities and features**
- **Design of Proxy**
- **Why Node.js?**
- **Performance**
- **Node.js weaknesses**
- **Used module and experiences**

What's the problem

- MySQL is still most reliable and popular in large scale applications
- Sharding of data is tightly-coupled with MySQL server deployment
- Resources waste in small applications
- Inefficient Ops with mass MySQL servers

Platform Architecture



Proxy functionalities and features

- Totally compatible with MySQL network protocol
- Sharding transparent to applications
- Separation of read and write
- Load balance
 - Random select read server, weight supported
 - Hash by IP
- DB failure decision by Multi node, auto switchover

Design of Proxy

- Sharding transparent to applications
 - Map rowkey to N tables
 - Divide N to M ranges, each map to a DB group(1master /n slaves)
 - Divide each range when expanding
- MySQL long connection pool
 - Pool for each Database
 - Init connection params by configuration
- Don't parse data of MySQL response, only transfer

Design of Proxy

- High performance and simple SQL Parser(C++)
- Transaction
 - Weak support, only single rowkey
- Seamless reload conf
- Abort C/S connections when exception
- Auto restart worker process when abort accidentally or no responses
 - Cluster module
 - Heart beat between master and worker

Why Node.js?

- High Performance for highly concurrent I/O
- Interface of Javascript, heart of C++
- Functional Programming & Closure
- Everything Non-block
- Low memory consumption
- No Lock
- Rich modules

Performance

- Environment:
 - CPU: Intel(R) Xeon(R) E5620 @ 2.40GHz
 - Cores: 2*4
 - Memory: 32GB
 - Linux kernel: 2.6.30
 - node.js version: 0.6
 - Multi-processes: 8
- QPS(Client Long Connection)
 - Access virtual table: 45000
 - Access non virtual table: 62000
- Resource consumption
 - cpu idle: 0%, mem: 50MB*8(processes), txkB/s: 20k, txpck/s: 110k
- Stability
 - No coredump, No memleak

Concurrency

- Support 1k concurrency with no more than 100MB memory

Concurrency	QPS(1 process)
10	8200
250	7800
500	7300
1000	7200

Node.js weaknesses

- Too flexible
 - JavaScript: The Good Parts (Douglas Crockford)
 - “use strict”
- No friendly encapsulation
- Nested callbacks
 - Async module(synchronous style code)
- Exception handling in callbacks
 - Domain(node v0.8)
- Multi processes load balance in cluster
 - assign connections to less processes by OS
 - will be improved in next version
- Unstable API and modules
- Possibly poor performance in large memory cases

Pains

- Memleak
 - setInterval/setTimeout must be cleared
- Use Buffer instead of binary string
- Pass Buffer Object to C++ extension is allowed
- Object values are references
- In ‘uncaughtException’, print err.stack
- “debugger;” ruin performance

Used modules

- underscore
- async
- log4js
- node-webkit-agent
- mocha
- node-zookeeper
- head_body_buffers

Used modules – log4js

- Filter high level logs to seperated file
- Not perfect in Multi-processes
- Take lot of memory when disk full

```
{  
  "appenders": [  
    {  
      "category": "maya",  
      "type": "file",  
      "filename": "../log/maya.log",  
      "maxLogSize": 2048000000,  
      "backups": 8,  
      "layout": {  
        "type": "basic"  
      }  
    },  
    {  
      "levels": {  
        "maya": "INFO"  
      }  
    }  
  ]  
}
```

Used modules - node-webkit-agent

- MUST set env of DEBUG_HOST to be real IP when debugging remote

The screenshot shows the Chrome DevTools interface with the 'Profiles' tab selected. On the left, there's a sidebar with sections for 'CPU PROFILES' and 'HEAP SNAPSHOTS'. Under 'CPU PROFILES', a profile named 'org.nodejsprofiles.cpu.user-initiate...' is selected. The main area displays a table of CPU usage statistics:

	Self	Total	Function	
	15.48s	15.48s	(program)	
	779ms	779ms	(garbage collector)	
	86ms	86ms	parse	
	31ms	31ms	execute	
	9ms	19ms	▶ afterWrite	
	4ms	1.03s	▶ startup.processNextTick.process._tickCallback	
	2ms	11.16s	▶ onread	
	1ms	1ms	exports.Db.pkFactory	/User
	1ms	1ms	BSON.deserialize.readCString	/User
	1ms	1ms	afterConnect	
	1ms	1ms	exports.active	
	0	2ms	▼ onconnection	
	0	1ms	▶ Socket	
	0	1ms	▶ EventEmitter.emit	

Used modules - node-webkit-agent

The screenshot shows the Chrome DevTools Profiles tab with the following details:

Elements Resources Network Scripts Timeline Profiles Audits Console

Search Profiles

Profiles

CPU PROFILES

- org.nodejs.profiles.cpu.user-initiated.1
- org.nodejs.profiles.cpu.user-initiated.2

HEAP SNAPSHOTS

- org.nodejs.profiles.heap.user-initiated.1 11.71MB
- org.nodejs.profiles.heap.user-initiated.2 11.84MB
- org.nodejs.profiles.heap.user-initiated.3 11.79MB
- org.nodejs.profiles.heap.user-initiat... 11.87MB

Class filter

	# New	# Deleted	# Delta	Alloc. Size	Free'd Size	Size Delta		
Constructor								
▶ (array)	634	308	+326	151 096	59 864	+91 232		
▶ (string)	267	152	+115	11 000	6 112	+4 888		
▶ (compiled code)	91	16	+75	97 200	30 368	+66 832		
▶ (system)	96	24	+72	4 552	1 104	+3 448		
▶ Object	76	21	+55	2 192	624	+1 568		
▶ (closure)	136	90	+46	9 792	6 480	+3 312		
▶ Array	45	28	+17	1 456	912	+544		
▶ Element	4	0	+4	224	0	+224		
▶ (number)	14	12	+2	224	192	+32		
▶ (regexp)	1	0	+1	72	0	+72		
▶ Buffer	9	8	+1	504	448	+56		
▶ Client	1	0	+1	24	0	+24		
▶ Date	8	7	+1	256	224	+32		
▶ Engine	1	0	+1	168	0	+168		
▶ IncomingMessage	2	1	+1	352	176	+176		
▶ JID	1	0	+1	48	0	+48		
▶ ServerResponse	1	0	+1	112	0	+112		
▶ SlowBuffer	5	4	+1	160	128	+32		
▶ Socket	8	7	+1	1 024	896	+128		
▶ StreamParser	1	0	+1	24	0	+24		
▶ XMPP	1	0	+1	112	0	+112		
▶ delete	1	0	+1	24	0	+24		
Object's retaining tree								
Object				Shallow Size	Retained Size	Distance		
content in (map descriptors) [] @154239				1 392	0 %	212 696	2 %	5

Comparison: org.nodejs.profiles.heap.user-initiated.1

Used modules - domain

```
process.on('uncaughtException', function(err) {
  console.log("exception in uncaughtException:", err);
});

try {
  process.nextTick(function () {
    throw new Error("exception in nextTick callback");
  });
} catch (err) {
  console.log("exception in catch:", err);
}
```

Used modules - domain

```
var domain = require('domain');
var events = require('events');

var d = domain.create();
d.on('error', function(err) {
  console.log("error in domain:", err);
});

var e;
d.run(function () {
  e = new events.EventEmitter();
});

e.on('msg', function () {
  throw new Error("exception in msg callback");
});

e.emit('error', new Error("error msg"));
e.emit('msg');
```

Used modules – head_body_buffers

- socket.on('data'), buffer no more than 64KB
- mysql packet
 - head(4 bytes) + body(n bytes)

```
function packetLength(data) {  
    return data[0] + (data[1] << 8) + (data[2] << 16);  
}
```

```
var hbd = new HeadBodyBuffers(4, packetLength);  
var client = net.connect(3306);  
client.on('data', function(data) {  
    hbd.addBuffer(data);  
});  
  
hbd.on('packet', function (packet) {  
    var head = packet.slice(0, 4);  
    var body = packet.slice(4);  
});
```

Summary

- Node.js reduce the difficulty of no-block network programming
- Good balance for performance and programming
- Lots of participations make nice environments
- Still young

THANKS