How to provide a reliable ridesharing service

DiDi Chuxing company service reliability assurance team

Ming Hua:  huaming@didichuxing.com
Lin Tan:  tanlintano@didichuxing.com
About Speakers

• **Ming Hua**:  
  • Principal Architect, SRE team, DiDi Chuxing, China  
  • Areas of focus:  
    • Service reliability  
    • Operation automation  
  • Email: huaming@didichuxing.com

• **Lin Tan**:  
  • Senior Engineer, SRE team, DiDi Chuxing, China  
  • Areas of focus:  
    • Service reliability  
    • Container and cloud platform  
  • Email: tanlintano@didichuxing.com
Agenda

1. Introduction of DiDi Chuxing
2. The challenges of reliability construction in rapidly growing service
3. Our technical solutions on service reliability
4. Our competition mechanism for reliability work
5. What we learned
Introduction of DiDi Chuxing

• **Founded in 2012 in China**
  • Experienced about 5 years rapid growth
  • One of the most fastest growing companies in the world.

• **The world’s leading mobile transportation platform**
  • 400+ million users
  • 17+ million drivers
  • 400+ cities

• **DiDi Chuxing Services**
  • Taxi
  • Premier car
  • Express
  • Hitch
  • Chauffeur
  • Minibus
  • Bus
  • Test Drive
  • Enterprise
  • Car Rental
Challenges of service reliability in DiDi Chuxing

- **System overload**: 500% annual requests growth
- **Release risk**: more than 400 releases per day without standard process and environment
- **Vulnerable infrastructure**: Single-cluster architecture
- **Lack of stability improvement**: Missing a measurable mechanism
Our Solutions

**Full-link stress test**
A specific ridesharing service stress test for system capacity evaluation and bottleneck location.

**Standardization**
The standardization practices on configuration management, service monitor, and program delivery.

**Multi-cluster service**
A high availability system which across different region zones, and a load balancing strategy based on cities.

**Competition Mechanism**
A special mechanism for system reliability and all the works of the team.
Full-link stress test
Full-link stress test

• **Purpose:**
  - Service capacity evaluation
  - Bottleneck location

• **Challenges:**
  - Data separation
  - Data accuracy

• **Strategy:**
  - Simulation
  - online test
Stress test - strategies for simulation

- Virtual country of China in the Pacific Ocean
- Virtual passengers send virtual orders
- Virtual drivers take virtual orders

Virtual country = original coordinate + offset + map information
Virtual passenger/driver = original id + offset + passengers/drivers information (some attributions were also made offset)
Stress test - virtual data logic isolation

Reasonable offset and range make sure don’t overlap with any real data

<table>
<thead>
<tr>
<th>Virtual data</th>
<th>construction</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>City ID mapping</td>
<td>ID + 10000</td>
<td>Not overlap with any normal city ID</td>
</tr>
<tr>
<td>Coordinate mapping</td>
<td>Lng - 230.80078&lt;br&gt;Lat - 59.63827</td>
<td>Locate in Pacific Ocean&lt;br&gt;Not overlap with any normal country</td>
</tr>
<tr>
<td>Passenger and driver ID</td>
<td>ID + 140737488355328</td>
<td></td>
</tr>
<tr>
<td>Virtual passenger phone number</td>
<td>111000020000-111699999999</td>
<td>About 68 million</td>
</tr>
<tr>
<td>Virtual driver phone number</td>
<td>111700000000-111999999999</td>
<td>About 30 million</td>
</tr>
<tr>
<td>Virtual order ID</td>
<td>High 8 bits is non 0</td>
<td>Normal id range is enough to use for million years</td>
</tr>
</tbody>
</table>
## Stress test - tag for test traffic

<table>
<thead>
<tr>
<th>HTTP communication</th>
<th>Thrift communication</th>
<th>Database request</th>
<th>Business traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add <strong>hint-code=1</strong> to http header</td>
<td>Add <strong>hintCode</strong> to message structure</td>
<td>Add <strong>hint tag</strong> before sql</td>
<td>Distinguish by <strong>specific range</strong></td>
</tr>
</tbody>
</table>

**HTTP communication**
- didi-header-rid:$traceid;
- didi-header-spanid:$spanid;
- didi-header-hint-code:$hintCode;
- didi-header-hint-content:$hintContent;

**Thrift communication**
- struct trace {
  1: required string logid;
  2: required string caller;
  3: optional string spanid;
  4: optional string srcMethod;
  5: optional i64 **hintCode**;
  6: optional string hintContent;
}

**Database request**

```sql
/*{"mode":"shadow"}*/
$sql
```

**Business traffic**
- cityid;
- passengerid;
- driverid;
- phone;
- orderid;
Stress test - data storage and cleaning

Mysql/rockstable
- Add 「hint」before sql. dbproxy makes detection and distribution
  - Business sql
  - dbproxy
- Select * from order
- original talbe (order)
- shadow talbe (order_shadow)

Persistent storage

Cache storage

Log

Different strategies for different storage types

Business log
- Add 「hint」tag to program log
  - Eg.: hint=1;

Codis
- Automatically clean data by set a short value for TTL (less than 30 minutes)
  - Kafka/beanstalkd
- Write to shadow tube/topic
  - Eg.: tube -> tube_shadow

Queue
Stress test - monitor support

- Log: Distinguish by tag
- Metrics: Program report in different tag
- DB: Read from shadow table

Monitor System
Stress test - traffic generation

Online stress test and implement during low peak period of the business

- Virtual customer orders: tens of thousands per minute
- Simultaneous online drivers: several millions
Standardization
Standardization

• The benefits of a standardization environment

  - Stability
  - Efficiency

  standardization

• The necessary of standardization in DiDi (3H)
  ○ Hard to handle service relationship
  ○ Hard to do trouble-shooting
  ○ High risks of misoperation and delivery accident
Our Standardization works

Focus on three fields:

- configuration
- monitor
- delivery
Standardization for Configuration

Connection relationship management standardization - disf

- Relay on IP: hard to manage and change
- Relay on NS: centralized management and easy to change
Standardization for Monitor

Principle: all programs involve metrics-lib and report their own status

Every business have a health dashboard
Standardization for Delivery

**Man dependent => fully-automatic workflow:**
- blue/green deployment
- necessary pause in every step
- standard deployment path and backup
- standard program control interface: `control.sh start|stop|reload`

**Semi-automatic**

**Fully-automatic workflow**

- launch → compile → build
- A module description file in json goes through the whole flow
- full release → gray release → Preview release
- monitor
Result

• Frequent service crash caused by misoperation and delivery have been well controled.

• Accident alarm and fault location takes less time.
Multi-cluster service
Multi-clusters service

It often takes a long time to recover even a slight fault occurs

All requests of 400+ cities

IDC TOR Failures
IDC Network Failures
Service Failures
Other Failures

Sign in  Sign up
Request  Respond
Dispatch  Bid
Pay  Review

There is no backup IDC for traffic switching
Multi-clusters challenges

• The difficulties of building multi-clusters for ridesharing service
  o Most of modules are stateful;
  o Consistency requirement for data;
  o Both driver-client and passenger-client are location-aware;
Multi-cluster implementation

All factors related to a region are closed to one unit/cluster

Requests of passengers and drivers from all the cities

Key solutions:

- All data of drivers and passengers and others was distinguished by city, and was split to more than one copies.
- Each cluster has partial data and a copy of other cluster.
- Dispatching requests by cityID in the request entrance.
Competition Mechanism
The Principal Rule

If You Can't Measure It, You Can't Improve It

(William Thomson, Lord Kelvin)
Our competition mechanism for reliability work

A competition named 「starflower」: A mechanism for ensuring all teams to invest in reliability work
Starflower - Competition Rules

**Judge:** Organizing Committee (OC)

**Participants:** All the reliability related teams (across nine business units and the technical departments)

**Goal:**
- Server reliability:
  - 2016: 99.9% downtime per month less than 40 minutes.
  - 2017: 99.95% downtime per month less than 22 minutes.

**Rule:**
- Downtime:
  - The time when the core performance indicators (request, respond, pay) was decreased by 10% * the affect orders / the total number of weekly orders.
- Quota:
  - The downtime available was assigned to each participants as their reliability quota.
**Starflower - exception handling**

**Core Indicators monitoring**: Phone call in urgent case

**Firemen Group**: Communication channel

**Fire Map**
Starflower - case follow

Case study:
- Case review
- Issue Analysis
- Improvement Analysis
- Duty review
Starflower - Rewards and punishments

Failure level:
• p1-p5 according to the severity of the failure

Punishment:
• 1,0000 RMB (about $1400) penalty for P1
• Summary conference every month
  • The teams who held their quota and done well jobs in reliability will be rewarded in money and honor titles;
  • The teams who broke their quota will be punished by money and negative titles;
• CEO, CTO or other VPS will be the presenter;
Result and the key points

All the teams and the whole technical department regard stability as the most important work

Results:
✓ 99.9% 2016
✓ 99.95% 2017 (up to now)

Key points:
✓ The judge should not be the competitor
✓ Measurable
✓ Clear rules
✓ Punishment and reward
✓ Continuous follow-up

Results:
99.9% 2016
99.95% 2017 (up to now)
What we learned
What we learned

The early the better  Collaborative working  Long term investment
Thanks!