分布式赛道

智能路由赛题详述

一．题目内容

在给定节点总数(N)，通道参数(C,M,delay,count)的条件下，设计一个网络的动态拓扑结构和对应的路由规则，并能同时满足银行间数据传输速率及稳定性的要求。这是一个分布式的集群，没有中央节点，需要自协同地实现通道规划及消息调度。

1. 参数描述：

1）有 N 个节点，第 i 个节点可以最大同时保持C_i个通道，初始条件下，节点之间没有通道，无法传递任何信息。

2） 最多有M_h个高速通道，每条通道的延迟为$delay_h$，最大允许$count_h$条消息在通道上传输，建立一条高速通道的耗时为$build_h$，销毁不需要时间。

3）最多有M_l个低速通道，每条通道的延迟为$delay_l$，最大允许$count_l$条消息在通道上传输，建立一条低速通道的耗时为$build_l$，销毁不需要时间。

2. 数据范围：

$10 < \mathrm{N} < 1000$
$\frac{N}{2} < M_h < N \times 4$,
$\mathrm{N} \times 3 < M_l < N \times 6$
$3 \le C_i \le log_2^N$
$0.3\mathrm{s} \le \mathrm{build} \le 1.8\mathrm{s}$
$10\mathrm{ms} \le delay_h \le delay_l \le 500\mathrm{ms}$
$4 \le \mathrm{count} \le 20$

二．规则说明及示例

1. 通道建立规则：

1）初始集群中节点相互孤立，没有任何通道

2）通道建立流程：

a）任意时刻(T_0)，任意节点(A)可以向其他任意节点(B)申请建立通道。

b）通道建立满足要求(满足 M 限制和 C 限制)，在一半创建时间之后($T_1 = T_0 + \frac{delay}{2}$)，B 节点会收到通道创建请求。

c）B 节点可以根据自身状态自行决定是否接受通道创建，并发送响应消息，若在 delay 时间内未发送任何消息，A 节点将收到 B 节点响应超时的消息($T_1 + delay$时间)。

d）若 B 节点接受建立通道，并在某个时刻($T_2, T_1 < T_2 < T_1 + delay$)返回接受消息，且通道建立满足要求，A、B 将在$T_2 + \frac{delay}{2}$时刻收到通道创建成功的消息，消息中包含该通道的通道号(channelId)。此后 A、B 之间可以通过该条通道传输消息。

e）若 B 节点拒绝建立通道，并在某个时刻($T_2, T_1 < T_2 < T_1 + delay$)返回拒绝消息，A 节点将在$T_2 + \frac{delay}{2}$时刻收到通道创建被拒绝的消息。

3）通道销毁流程：

对于任意两个节点(A、B)，若 A、B 之间存在一条通道，则在任意时刻(T_0)，A 可以向 B 发送通道销毁的消息，B 将立即收到通道销毁协议(T_0)，此后，该通道不再可用。

4）通道的实现由主办方提供，直接调用 api 即可。(主办方提供 channel.h、channel.jar、

channel.py，各语言接口如下图所示)

```java
import json.Config;
import json.Message;

import java.util.List;

public interface Channel {

    /**
     * 初始化管道
     * @param config
     */
    void initConfig(Config config);

    /**
     * 将消息发送给指定目标
     * @param message
     * @param targetId
     */
    void send(Message message, int targetId);

    /**
     * 获取缓存中的所有消息
     * @return 消息列表
     */
    List<Message> recv();

    /**
     * 获取当前节点编号
     * @return 获取当前节点编号
     */
    int getId();
}
```

```cpp
#pragma once
#include <vector>
#include "extra/json.hpp"

using json = nlohmann::json;
class Channel {
public:

    /**
     * 初始化管道
     * @param config
     */
    virtual void initConfig(json config);

    /**
     * 将消息发送给指定目标
     * @param message
     * @param targetId
     */
    virtual void send(json message, int targetId);

    /**
     * 获取缓存中的所有消息
     * @return 消息列表
     */
    virtual std::vector<json> recv();

    /**
     * 获取当前节点编号
     * @return 当前节点编号
     */
    virtual int getId();
};
```

```python
class Channel(object):

    def init_config(self, config):
        """
        初始化管道
        :param: config
        """

    def send(self, message, target_id):
        """
        将消息发送给指定目标
        :param: message
        :param: target_id
        """

    def recv(self):
        """
        获取缓存中的所有消息
        :return: 消息列表
        """

    def get_id(self):
        """
        获取当前节点编号
        :return: 当前节点编号
        """

```

2. **测试相关规则**

1）在每个测试样例中，测试机会在任意时间随机选择两个节点(A,B)，向其中一个节点(A)发送 prepare 消息，在间隔 delay(其中 0.2s<delay<2.6s)时间之后，再向同一个节点(A)发送 send 消息(data 为一个 256 位 hash int)，若在指定时间内(记为 σ)从另一个节点(B)收到包含相同 data 的消息，即认为这条消息送达，否则认为消息超时。

2）单组测试内包含多个测试样例，任意两条 prepare 消息的发送间隔大于 0.1s。

3）单组测试内通道参数(通道数M_h和M_l、通道传输时延、通道创建时延、通道容量)、节点总数(N)、延迟限制(σ)在初始化时提供，并在测试过程中不变。

4）得分仅与 send 消息的时延(记为 τ)以及 send 消息的送达率(记为η)有关。未送达的消息时延不计入 τ 的统计，而计入 η 的统计。时延的均值记为$\bar{\tau}$，方差记为D_τ，

则该组测试的得分为：

$$G=(\frac{\sigma\times 2-\bar{\tau}}{\sigma}\times 3.5+\frac{0.5}{D_\tau+1})\times \eta^{2.5}\times 10$$

5）包含多组测试数据，总分为各组测试得分之和。即总分为$\sum G$。

3. 消息定义

```
{
  "callType": "prepare",
  /*
      CALL_TYPE_PREPARE = "prepare"
      CALL_TYPE_SEND = "send"
      CALL_TYPE_SYS = "sys"
      CALL_TYPE_CHANNEL_BUILD = "channel_build"
      CALL_TYPE_CHANNEL_DESTROY = "channel_destroy"
  */
  "channelId": 40, // 经由通道编号
  "sysMessage": {
    "target": 40, // 消息目的地
    "data": "anything", // 消息传输数据内容
    "delay": 2.5 // 消息延迟到达时间
  },
  "extMessage": {}, // 自定义消息内容,只限制长度,不限制内部具体格式
  "state": 0,
  /*
      STATE_REQUEST = 0
      STATE_ACCEPT = 1
      STATE_REFUSE = 2
      STATE_NOTICE = -1
  */
  "errCode": 0,
  "channelType": 0
  /*
      CHANNEL_TYPE_NORMAL = 0
      CHANNEL_TYPE_FAST = 1
  */
}
```

如上图 json 所示，是 message 的全量定义，用以描述一封消息。message 分为五种不同的 callType，不同类型的 message 所含的字段及含义亦不相同。消息分为预备消息(prepare)、传输消息(send)、自定义消息(sys)、通道创建消息(channel_build)、通道销毁(channel_destroy)消息五类。

各类消息相应字段有效性见下表：

字段 / 消息	callType	channelId	sysMessage	extMessage	state	errCode	channelType
预备消息	有效	有效，指定传输通道	data 字段无效	有效，默认为空	有效	有效	无效，通道类型由 channelId 指定

传输消息	有效	有效，指定传输通道	delay 字段无效	有效，默认为空	有效	有效	无效，通道类型由 channelId 指定
自定义消息	有效	有效，指定传输通道	无效	有效，默认为空	有效	有效	无效，通道类型由 channelId 指定
通道创建	有效	有效，指定作用通道	target 字段有效	无效	有效	有效	有效
通道销毁	有效	有效，指定作用通道	无效	无效	有效	有效	无效

4. 消息流程及示例：

1) 预备消息、传输消息、自定义消息示例：

```
{
  "callType": "prepare",
  "channelId": 0,
  "sysMessage": {
    "target": 40,
    "data": "",
    "delay": 2.5
  },
  "extMessage": {},
  "state": -1,
  "errCode": 0,
  "channelType": 0
}
```

```
{
  "callType": "send",
  "channelId": 0,
  "sysMessage": {
    "target": 40,
    "data": "anydata",
    "delay": 0
  },
  "extMessage": {},
  "state": -1,
  "errCode": 0,
  "channelType": 0
}
```

对于上面的预备消息，描述了在 2.5 秒之后，会有一条 send 消息发往 40 号节点。在 2.5 秒之后，send 消息到达，描述了有一条数据”anydata”需要传输到 40 号节点。

```
{
  "callType": "sys",
  "channelId": 70,
  "sysMessage": {
    "target": 0,
    "data": "",
    "delay": 0
  },
  "extMessage": {
  | "routingTable":[...]
  },
  "state": -1,
  "errCode": 0,
  "channelType": 0
}
```

对于 sys 消息，这种消息发生于系统内部，用于节点之间交换数据，在上述例子中，这条消息将通过 70 号通道传递，内容是一张路由表。

在上述三类消息中，extMessage 字段是一个可选字段，节点可以任意修改其中的值。

2) 对于通道创建，一共分为如下 5 步：

此处假设节点 16 希望创建一条连接节点 38 的低速通道。 因为通道尚未创建成功，因此所有 channel_build 相关的 message 发送时的 targetId 都是 0，因此这封 message 并不是直接发送给通道另一端节点。创建流程如下：

1、16 号节点向集群申请创建一条低速通道，连向 38 号节点。

```
{ //1
  "callType": "channel_build",
  "channelId": 0,
  "sysMessage": {
    "target": 38,
    "data": "",
    "delay": 0
  },
  "extMessage": {},
  "state": 0,
  "errCode": 0,
  "channelType": 0
}
```

2、38 号节点收到来自集群的通知，内容为 16 号节点需要创建一条低速通道。

```
{ //2
  "callType": "channel_build",
  "channelId": 0,
  "sysMessage": {
    "target": 16,
    "data": "",
    "delay": 0
  },
  "extMessage": {},
  "state": -1,
  "errCode": 0,
  "channelType": 0
}
```

3、38 号节点同意来自 16 号节点的请求。

4、集群同意该通道的创建，分配通道编号为 203，发送给 16 号节点。

5、集群同时将通道创建成功的消息通知为 38 号节点。

3）以下事件将使得通道创建失败：

a) 在第一封消息发送之后，集群检测到创建此通道不满足网络要求。

b) 在第二封消息收到后，38 号节点**主动**拒绝创建通道。

c) 在第二封消息收到后，38 号节点响应超时。

d) 在第三封消息发出后，集群检测到创建此通道不满足网络要求。

在创建通道失败后，集群会发送拒绝创建通道的消息给通道双方，如下图所示。

```
{ //6
  "callType": "channel_build",
  "channelId": 0,
  "sysMessage": {
    "target":38,
    "data": "",
    "delay": 0
  },
  "extMessage": {},
  "state": 2,
  "errCode": 0x103,
  "channelType": 0
}
```

4）集群的检测逻辑在测试机上实现

5）对于通道销毁协议，targetId 依然是 0，只有两步：

a) 通道一端的节点向集群发起请求，申请销毁 177 号通道。

b) 集群响应请求，并通知通道相连的两个节点。

```
{ //1
  "callType": "channel_destroy",
  "channelId": 177,
  "sysMessage": {
    "target": 0,
    "data": "",
    "delay": 0
  },
  "extMessage": {},
  "state": 0,
  "errCode": 0,
  "channelType": 0
}
```

```
{ //2
  "callType": "channel_destroy",
  "channelId": 177,
  "sysMessage": {
    "target": 0,
    "data": "",
    "delay": 0
  },
  "extMessage": {},
  "state": -1,
  "errCode": 0,
  "channelType": 0
}
```

6) 任意节点只能操作与自己相连的通道(通过通道发送消息或者销毁通道)，若操作的通道不存在或不与操作节点相连，集群会返回一个包含如下图所示的消息，其中 callType 字段与非法请求的消息相同。

```json
{
  "callType": "",
  "channelId": 0,
  "sysMessage": {
    "target": 0,
    "data": "",
    "delay": 0
  },
  "extMessage": {},
  "state": 2,
  "errCode": 0x001,
  "channelType": 0
}
```

7) 各 err 类型如下图所示：

```
ERR_CODE_NO_SUCH_CHANNEL = 0x001;
ERR_CODE_CHANNEL_BUILD_MASK = 0x0100;
ERR_CODE_CHANNEL_BUILD_TARGET_REFUSE = 0x0101;
ERR_CODE_CHANNEL_BUILD_TARGET_LIMIT = 0x0102;
ERR_CODE_CHANNEL_BUILD_TOTAL_LIMIT = 0x0103;
ERR_CODE_CHANNEL_BUILD_SOURCE_LIMIT = 0x0104;
ERR_CODE_CHANNEL_BUILD_TARGET_TIMEOUT = 0x0105;
ERR_CODE_SEND_MASK = 0x200;
ERR_CODE_SEND_COUNT_LIMIT = 0x201;
ERR_CODE_SEND_SIZE_LIMIT = 0x202;
```