دانشگاه صنعتی امیرکبیر
دانشکده مهندسی کامپیوتر و فناوری اطلاعات

سیستم‌های توزیعی
پروژه دوم

پرهام الوانی

۱ دی ۱۳۹۶

۱ فاز اول

برای پیاده‌سازی پروژه سه گام در نظر گرفته شده است. در اولین گام ارتباط بین سرور و کلاینت پیاده‌سازی شد و در گام ارتباط بدون خطا فرض می‌شود. در دومین و سومین گام به ترتیب ارتباط دارای خطا و بسته شدن صحیح سرور و کلاینت پیاده‌سازی گشت.

دو قسمت اصلی در کلاینت در نظر گرفته می‌شود، قسمت اول receiver است که داده‌ها را از سوکت دریافت کرده و در کانال incoming می‌ریزد. قسمت دوم handler نام دارد که در واقع قلب سیستم است و تمام کنترل‌ها در آنجا صورت می‌گیرد دلیل این امر جلوگیری از وقوع race condition می‌باشد.

handler داده‌ها را از کانال incoming دریافت می‌کند و آن‌ها را در یک hash map بر اساس شماره دنباله‌شان ذخیره می‌کند. در ادامه پیام‌های ack مناسب را تولید می‌کند. پیام‌هایی که ترتیب آن‌ها درست است در کانال rmsg برای دریافت توسط لایه‌ی بالاتر قرار می‌گیرند.
لایه اپلیکشن هم پیام‌هایی که می‌خواهد ارسال شوند را در کانال tmsg قرار می‌دهد. handler این پیام‌ها را نیز در یک hash map بر اساس شماره دنباله‌شان قرار می‌دهد با این تفاوت که طول این hash map هرگز نباید از window size (windows) بزرگتر شود. با دریافت هر ack آن پیام از لیست پیام‌های در حال ارسال حذف می‌شود.

handler با رویدادهای زمانی که در بازه‌های زمانی epochMillis رخ می‌دهند سه کار انجام می‌دهد. در صورتی که هیچ پیامی دریافت نشده باشد ack با شماره دنباله‌ی صفر ارسال می‌کند و اگر داده‌هایی در حال ارسال داشته باشد تمام آن‌ها را ارسال می‌کند. در صورتی که هنوز ارتباط با سرور شکل نگرفته باشد پیام connect را باز تکرار می‌کند.
برای بستن ارتباط یک کلاینت دو حالت وجود دارد. در اولین حالت ارتباط با سرور timeout می‌کند که در این حالت ارتباط بدون توجه به وضعیت پیام‌ها بسته می‌شود و پیام خطا در کانال err قرار می‌گیرد. در دومین حالت ارتباط به درخواست کاربر بسته می‌شود که در این حالت نیاز است که handler تمام پیام‌هایی که در صف بافی مانده است را ارسال کند. برای مدیریت این شرایط از متغیر status استفاده می‌کنیم status در واقع یک مانیتور است که دسترسی به یک integer را مدیریت می‌کند.

در سمت سرور هم از همان ایده‌ی کلاینت استفاده شده است فقط در اینجا نیاز به دو handler داریم یکی برای سرور و دیگری به ازای هر کلاینت وجود خواهد داشت. receiver نیز مانند سابق پیام‌ها را دریافت کرده و در کانال incoming قرار می‌دهد.
رفتار handler هر کلاینت مانند زمانی است که در کلاینت اجرا شده است با این تفاوت که از یک کانال broadcast برای مدیریت بستن هر کلاینت استفاده می‌شود. کانال‌های broadcast در go کانال خالی هستند که در زمان وقوع یک رویداد آن‌ها را می‌بندیم در زمانی که یک کانال بسته می‌شود تمام کسانی که در حال دریافت از آن‌ها هستند مقدار صفر از آن کانال می‌خوانند، به این ترتیب می‌توانیم تعداد زیادی goroutine را از یک رویداد مطلع سازیم.
هر کلاینت می‌بایست بتواند خطا و داده‌ی خود را به لایه‌ی اپلیکشن منتقل کند. برای جلوگیری از race condition کانال err و rmsg به صورت global روی سرور تعریف شده‌اند و همه‌ی کلاینت‌ها داده‌های خود را به همراه شناسه‌شان در آن‌ها قرار می‌دهند.
برای دریافت داده‌های لایه‌ی اپلیکشن و جلوگیری از بروز race condition نیز یک کانال outgoing بر روی سرور استفاده می‌کنیم که داده‌ها را از لایه‌ی اپلیکشن دریافت کرده و آن‌ها را در یک کانال tmsg کلاینت مورد نظر قرار می‌دهد.

۲ فاز دوم

در این فاز با استفاده از پروتکل lsp که در فاز قبل پیاده‌سازی شد یک distributed bitcoin miner پیاده‌سازی شد.
این فاز از ۳ فایل اصلی تشکیل می‌شود:

- client.go وظیفه‌ی آن ارسال تقاضا می‌باشد
- server.go وظیفه‌ی مرتب کردن تقاضاها، تخصیص آن‌ها و زمان‌بندی را بر عهده دارد.
- miner.go محاسبات hash را در بازه‌ی داده شده انجام می‌دهد.

سرور تقاضاها را دریافت می‌کند، از آنجایی که ممکن است زمان دریافت یک تقاضا هیچ minerای موجود نباشد به صورت زیر هر تقاضا به تعدادی زیر تقاضا شکسته شده و در نهایت در صف قرار می‌گیرد.

Listing 1: Partioning Algorithm

```
var stride = 1
if r.upper > 10000 {
        stride = int(math.Floor(math.Log10(float64(r.upper))))
}
if srv.freeMiners.Len() > 0 {
        if stride < srv.freeMiners.Len() {
                stride = srv.freeMiners.Len()
        }
}
```

در زمانی که پاسخ هر تقاضا داده می‌شود یا یک miner جدید به سیستم اضافه می‌شود این miner به تقاضاهایی که در صف قرار دارند تخصیص داده می‌شود در این تخصیص معیار اولویت ترتیب در صف است پس برای تقاضاهای duplicate که می‌خواهیم اولویت بیشتری داشته باشند آن‌ها را در ابتدای صف قرار می‌دهیم.
برای مدیریت بهتر اجرای زمان‌بند، از کانال trigger استفاده می‌کنیم، این کانال در واقع تعداد بارهایی که قرار است زمان‌بند اجرا شود را نگهداری می‌کند و هربار که قرار است زمان‌بند اجرا شود یک شی در آن قرار می‌گیرد.

تمام قسمت‌های اصلی سرور در قالب یک حلقه و select پیاده‌سازی شده است. دلیل این امر کنترل دسترسی به متغیرهای مشترک و جلوگیری از race condition می‌باشد.

پیاده‌سازی miner به وسیله‌ی goroutineها صورت گرفته است. به این ترتیب که برای هر تقاضا تعدادی goroutine ساخته می‌شود.

پیاده‌سازی client بسیار سرراست می‌باشد به این ترتیب که فقط منتظر پاسخ یا به پایان رسیدن timeout می‌شود. برای کنترل همزمان دریافت پاسخ و زمان‌سنج از select استفاده می‌شود.