گزارش پروژه سیستم‌های توزیع‌شده

پروژه سوم: پیاده‌سازی الگوریتم اجماع Raft

اعضای تیم:
محمد سعید صدیقی - ۸۱۰۱۰۰۱۷۹
محمد حسین مطاعی - ۸۱۰۱۹۹۴۹۳
محمد مهدی خصالی - ۸۱۰۱۰۰۱۳۴

دانشگاه تهران - دانشکده مهندسی کامپیوتر
بهار ۱۴۰۴

فهرست مطالب

۱ مقدمه **۲**

۲ معماری سیستم Raft **۲**
۱.۲ اجزای کلیدی Raft ۲
۲.۲ مفاهیم Term و Entry Log ۳

۳ پیاده‌سازی انتخاب رهبر (بخش ۳A) **۳**
۱.۳ توضیح مکانیزم انتخاب رهبر و نقش هاHeartbeat ۴
۲.۳ جزئیات پیاده‌سازی هایRequestVote RPC و AppendEntries ۴
۳.۳ نکات مهم پیاده‌سازی ۴

۴ پیاده‌سازی ثبت لاگ (بخش ۳B) **۵**
۱.۴ فرآیند افزودن و تکرار ورودی‌های لاگ ۵
۲.۴ نقش تابع ()Start و کانال applyCh ۵
۳.۴ محدودیت‌های رای‌گیری و تضمین سازگاری لاگ ۵
۴.۴ نکات پیاده‌سازی ۶

۵ پیاده‌سازی پایداری (بخش ۳C) **۶**
۱.۵ اهمیت وضعیت پایدار و بازیابی پس از راه‌اندازی مجدد ۶
۲.۵ استفاده از شیء Persister و توابع Save/ReadRaftState ۶
۳.۵ سریال‌سازی وضعیت با labgob ۷
۴.۵ الگوریتم بازگرداندن nextIndex (Backtracking nextIndex) ۷
۵.۵ نکات مهم ۷

۶ نتایج آزمایش‌ها **۷**
۱.۶ نتایج تست‌های انتخاب رهبر (بخش ۳A ۸
۲.۶ نتایج تست‌های ثبت لاگ (بخش ۳B ۸
۳.۶ نتایج تست‌های پایداری (بخش ۳C ۱۰
۴.۶ جدول ۱: خلاصه‌ای از نتایج تست‌ها ۱۱

۷ جزئیات و نکات پیاده‌سازی **۱۱**
۱.۷ مدیریت هم‌زمانی و Condition Race ۱۱
۲.۷ مدیریت خطا و عملیات دوباره ۱۱
۳.۷ بهینه‌سازی عملکرد ۱۲

۸ چالش‌ها و راه‌حل‌ها **۱۲**
۱.۸ چالش‌های کلیدی ۱۲

۹ نتیجه‌گیری **۱۳**

۱ مقدمه

این گزارش به تفصیل به پیاده‌سازی الگوریتم اجماع Raft در چارچوب یک پروژه درس سیستم‌های توزیع‌شده می‌پردازد. الگوریتم Raft یک پروتکل اجماع طراحی شده برای مدیریت لاگ‌های تکرار شده (Replicated Log) در سیستم‌های توزیع‌شده است. هدف اصلی این الگوریتم، فراهم آوردن تحمل‌پذیری در برابر خرابی (Fault Tolerance) و حفظ سازگاری (Consistency) داده‌ها در میان چندین سرور است، حتی زمانی که برخی از سرورها از کار می‌افتند یا شبکه ناپایدار می‌شود. Raft به عنوان جایگزینی قابل فهم‌تر برای الگوریتم‌های خانواده Paxos طراحی شده است. این قابلیت فهم بالا از طریق تفکیک منطق به اجزای مستقل‌تر مانند انتخاب رهبر، تکرار لاگ و تضمین‌های ایمنی حاصل می‌شود. با وجود سادگی بیشتر در درک، Raft از نظر کارایی با Paxos قابل مقایسه است و ایمنی آن به صورت رسمی اثبات شده است. این الگوریتم با سازماندهی درخواست‌های مشتری به صورت یک توالی مشخص به نام «لاگ» (Log) و اطمینان از اینکه تمامی سرورها نسخه‌ای یکسان از این لاگ را مشاهده می‌کنند، سازگاری را تضمین می‌کند. پروژه حاضر شامل پیاده‌سازی الگوریتم Raft به صورت یک شیء Go با توابع مرتبط است، به گونه‌ای که نمونه‌های مختلف Raft بتوانند از طریق RPC با یکدیگر ارتباط برقرار کرده و لاگ‌ها را هماهنگ نگه دارند. این پیاده‌سازی بر سه بخش کلیدی الگوریتم Raft متمرکز بوده است: انتخاب رهبر (Leader Election) که در بخش 3A مورد بررسی قرار گرفت، ثبت لاگ (Log Replication) که مربوط به بخش 3B است، و پایداری (Persistence) که در بخش 3C پیاده‌سازی شد. لازم به ذکر است که بخش فشرده‌سازی لاگ (Log Compaction) که به عنوان بخش 3D در دستورالعمل پروژه ذکر شده بود، در این پیاده‌سازی انجام نشده و بنابراین در این گزارش به آن اشاره‌ای نخواهد شد. انتخاب Raft برای این پروژه دانشگاهی در درس سیستم‌های توزیع‌شده، یک تصمیم آگاهانه آموزشی است. تأکید Raft بر قابل فهم بودن و تجزیه واضح مسائل اجماع (مانند انتخاب رهبر، تکرار لاگ و ایمنی) فرآیند یادگیری را برای دانشجویان تسهیل می‌کند و به آن‌ها کمک می‌کند تا مفاهیم پیچیده اجماع توزیع‌شده را به خوبی درک کنند. ما برای این پروژه از یک روز گریس استفاده کردیم.

۲ معماری سیستم Raft

الگوریتم Raft بر اساس مجموعه‌ای از سرورها بنا شده است که با همکاری یکدیگر، یک سرویس تکرار شده (Replicated Service) را ارائه می‌دهند. این سرویس با نگهداری نسخه‌های کامل از وضعیت داده‌ها در چندین سرور، تحمل‌پذیری در برابر خرابی را فراهم می‌آورد. در هسته معماری Raft، سه نقش اصلی برای سرورها تعریف شده است که هر سرور در هر لحظه در یکی از این حالت‌ها قرار دارد: رهبر (Leader)، پیرو (Follower) و کاندیدا (Candidate).

۱.۲ اجزای کلیدی Raft

- **سرورها و حالت‌ها:**

- **پیرو (Follower):** این حالت، وضعیت پیش‌فرض و منفعل یک سرور است. پیروها تنها به درخواست‌های رهبر یا کاندیداها پاسخ می‌دهند. اگر یک پیرو برای مدت زمان مشخصی که به آن «زمان‌بندی انتخابات» (election timeout) گفته می‌شود، هیچ پیامی از رهبر دریافت نکند، فرض می‌کند که رهبر از کار افتاده و وضعیت خود را به کاندیدا تغییر می‌دهد.
- **کاندیدا (Candidate):** سروری که از حالت پیرو به این حالت تغییر وضعیت داده است، به این معنی که قصد دارد رهبر جدید کلاستر شود. با آغاز یک دوره (Term) جدید، کاندیدا ابتدا به خودش رأی می‌دهد و سپس درخواست‌های رأی (RequestVote RPCs) را به سایر سرورها ارسال می‌کند تا حمایت آن‌ها را جلب کند.
- **رهبر (Leader):** تنها یک رهبر در هر کلاستر Raft در یک زمان مشخص وجود دارد. رهبر مسئول اصلی پذیرش درخواست‌های مشتری، تکرار ورودی‌های لاگ به تمامی پیروها، و

ارسال منظم پیام‌های ضربان قلب (Heartbeat) برای حفظ رهبری خود و جلوگیری از آغاز انتخابات جدید است.

- **گذار حالت (State Transition):**
  - **پیرو به کاندیدا:** زمانی اتفاق می‌افتد که زمان‌بندی انتخابات منقضی شود و هیچ پیام Heartbeat از رهبر دریافت نشود.
  - **کاندیدا به رهبر:** زمانی رخ می‌دهد که کاندیدا اکثریت آرا را از سرورهای دیگر کلاستر دریافت کند.
  - **کاندیدا به پیرو:** اگر کاندیدا یک پیام AppendEntries RPC یا RequestVote RPC با یک Term بالاتر از Term فعلی خود دریافت کند، بلافاصله به حالت پیرو بازمی‌گردد.
  - **رهبر به پیرو:** اگر رهبر یک RPC با Term بالاتر از Term فعلی خود دریافت کند، به حالت پیرو تغییر وضعیت می‌دهد.
- **لاگ (Log):** لاگ یک توالی مشخص از دستورات (Commands) است که تمامی سرورها باید نسخه‌ای یکسان از آن را داشته باشند. هر سرور دستورات را به ترتیب در لاگ خود اعمال می‌کند تا وضعیت یکسانی را در سراسر کلاستر حفظ کند.
- **RPCها (Remote Procedure Calls):** ارتباط بین نمونه‌های Raft به صورت انحصاری از طریق RPC انجام می‌شود و استفاده از متغیرهای اشتراکی بین نمونه‌ها مجاز نیست. دو نوع RPC اصلی در Raft وجود دارد:
  - RequestVote: توسط کاندیداها برای جمع‌آوری آرا در طول انتخابات ارسال می‌شود.
  - AppendEntries: توسط رهبر برای تکرار ورودی‌های لاگ و همچنین به عنوان پیام Heartbeat برای اعلام حضور خود به پیروها استفاده می‌شود.

۲.۲ مفاهیم Term و Entry Log

- **Term (دوره):** Term یک دوره زمانی منطقی و دلخواه در سرور است که با هر انتخابات رهبر جدید آغاز می‌شود. شماره Term به صورت یکنواخت افزایش می‌یابد و در تمامی ارتباطات RPC بین سرورها مبادله می‌شود. این مفهوم به Raft کمک می‌کند تا ناهماهنگی‌های زمانی را مدیریت کرده و یک ترتیب جهانی برای رویدادها فراهم کند. مکانیزم Term در Raft به عنوان یک ساعت منطقی حیاتی و یک مکانیزم ایمنی اصلی عمل می‌کند. افزایش یکنواخت آن و قانون بازگشت سرورها به حالت پیرو در صورت مشاهده Term بالاتر، به صورت بنیادی حل تعارضات را ساده کرده و سازگاری در سراسر کلاستر را تضمین می‌کند. این ویژگی، سنگ بنای مقاومت Raft در برابر تقسیم‌بندی شبکه و خرابی سرورها را تشکیل می‌دهد.
- **Log Entry (ورودی لاگ):** هر Log Entry شامل یک دستور (Command) است که باید توسط ماشین حالت اجرا شود، و همچنین Term مربوط به زمانی که این ورودی توسط رهبر دریافت شده است. ورودی‌ها به صورت اعدادی با شماره ایندکس (Index) ذخیره می‌شوند و هر ورودی معتبر باید وارد لاگ شده و سپس به سیستم بزرگتر تحویل داده شود.

۳ پیاده‌سازی انتخاب رهبر (بخش ۳(A

پیاده‌سازی بخش 3A بر مکانیزم انتخاب رهبر در الگوریتم Raft و همچنین پیاده‌سازی پیام‌های Heartbeat تمرکز دارد. این بخش برای تضمین وجود یک هماهنگ‌کننده فعال و پاسخگو در کلاستر حیاتی است.

۱.۳ توضیح مکانیزم انتخاب رهبر و نقش هاHeartbeat

فرآیند انتخاب رهبر در Raft به شرح زیر است: هنگامی که یک پیرو برای مدت زمان مشخصی که به آن «زمان‌بندی انتخابات» (election timeout) گفته می‌شود، هیچ پیامی از رهبر دریافت نمی‌کند، فرض می‌کند که رهبر فعلی از کار افتاده یا ارتباطش قطع شده است. در این حالت، پیرو وضعیت خود را به کاندیدا تغییر می‌دهد. کاندیدا Term خود را افزایش می‌دهد، به خودش رأی می‌دهد و سپس RPCهای RequestVote را به سایر سرورها در کلاستر ارسال می‌کند تا رأی آن‌ها را جلب کند. اگر کاندیدا بتواند اکثریت آرا را از سرورهای دیگر دریافت کند، به رهبر جدید کلاستر تبدیل می‌شود.

پس از انتخاب شدن، رهبر شروع به ارسال منظم پیام‌های Heartbeat به تمامی پیروهای خود می‌کند. این Heartbeatها در واقع پیام‌های AppendEntries هستند که محتوای لاگ خالی دارند. هدف اصلی این پیام‌ها، حفظ رهبری رهبر فعلی و جلوگیری از آغاز انتخابات‌های جدید توسط پیروها است. دریافت منظم Heartbeatها توسط پیروها، تایمر election timeout آن‌ها را بازنشانی می‌کند و از تغییر وضعیت آن‌ها به کاندیدا جلوگیری می‌کند.

۲.۳ جزئیات پیاده‌سازی هایRequestVote RPC و AppendEntries

پیاده‌سازی این بخش شامل تکمیل ساختارهای RequestVoteArgs و RequestVoteReply برای تبادل اطلاعات در طول فرآیند رأی‌گیری است. RequestVoteArgs شامل اطلاعاتی مانند Term کاندیدا، شناسه کاندیدا، ایندکس آخرین ورودی لاگ و Term آخرین ورودی لاگ است. RequestVoteReply نیز شامل پاسخ رأی‌دهنده (شامل Term فعلی رأی‌دهنده و اینکه آیا رأی داده شده است یا خیر) می‌باشد.

برای پیام‌های Heartbeat، ساختارهای AppendEntriesArgs و AppendEntriesReply مورد استفاده قرار می‌گیرند. AppendEntriesArgs شامل Term رهبر، شناسه رهبر، ایندکس و Term ورودی لاگ قبلی (PrevLogIndex, PrevLogTerm) برای بررسی سازگاری لاگ، و یک لیست خالی از ورودی‌ها (entries) برای Heartbeatها است.

۳.۳ نکات مهم پیاده‌سازی

- **تایمرهای انتخابات:** تنظیم بازه زمانی برای تایمرهای انتخابات بین ۱۵۰ تا ۳۰۰ میلی‌ثانیه توصیه می‌شود. استفاده از تایمرهای تصادفی (Randomized Election Timeout) برای هر سرور، به شدت احتمال برخورد و تقسیم آرا (Split Vote) را کاهش می‌دهد. این تصادفی‌سازی تضمین می‌کند که سرورها به صورت همزمان کاندیدا نمی‌شوند و یک سرور به احتمال زیاد قبل از اینکه دیگران زمان‌بندی‌شان منقضی شود، رهبر می‌شود.
- **مدیریت زمان:** برای پیاده‌سازی تایمرها، باید از time.Timer یا time.Ticker استفاده شود و از time.Sleep پرهیز شود.
- **مدیریت هم‌زمانی (Concurrency):** برای اطمینان از thread-safety و جلوگیری از تداخل دسترسی‌ها به وضعیت داخلی Raft، استفاده از sync.Mutex ضروری است.
- **عملکرد و زمان‌بندی تست:** برنامه تست انتظار دارد که رهبر جدید ظرف کمتر از پنج ثانیه انتخاب شود. پیاده‌سازی باید به گونه‌ای باشد که این شرط را برآورده کند.
- **ارتباط RPC-محور:** تمامی ارتباطات بین نمونه‌های Raft باید به صورت انحصاری از طریق RPC انجام شود و استفاده از متغیرهای اشتراکی بین نمونه‌ها مجاز نیست.

AppendEntries RPC در Raft یک عملکرد دوگانه حیاتی دارد: هم برای تکرار ورودی‌های لاگ و هم به عنوان سیگنال Heartbeat استفاده می‌شود. این طراحی هوشمندانه، سربار شبکه را به حداقل می‌رساند. زمانی که ورودی‌های لاگی برای تکرار وجود دارد، ارسال آن‌ها به طور ضمنی به عنوان Heartbeat عمل می‌کند. تنها زمانی که هیچ ورودی جدیدی برای تکرار نیست، رهبر یک AppendEntries RPC با

لاگ خالی ارسال می‌کند تا صرفاً نقش Heartbeat را ایفا کند. این رویکرد، ترافیک شبکه اضافی را کاهش می‌دهد و به سیستم کمک می‌کند تا الزامات زمانی سخت‌گیرانه تست‌ها (مانند انتخاب سریع رهبر) را برآورده کند، که به کارایی کلی سیستم در محیط توزیع‌شده کمک شایانی می‌کند.

۴ پیاده‌سازی ثبت لاگ (بخش ۳B)

بخش 3B بر پیاده‌سازی فرآیند اصلی تکرار لاگ در Raft تمرکز دارد، که شامل افزودن ورودی‌های جدید به لاگ، تکرار آن‌ها به پیروها و تضمین سازگاری در سراسر کلاستر است.

۱.۴ فرآیند افزودن و تکرار ورودی‌های لاگ

هنگامی که رهبر یک دستور جدید از کلاینت دریافت می‌کند، آن را به عنوان یک Log Entry به لاگ خود اضافه می‌کند. سپس رهبر این ورودی جدید را از طریق RPCهای AppendEntries به تمام پیروهای خود ارسال می‌کند. پیروها پس از دریافت و افزودن موفقیت‌آمیز ورودی به لاگ خود، تأییدیه‌ای (Acknowledgement) به رهبر ارسال می‌کنند. رهبر منتظر می‌ماند تا تأییدیه از اکثریت پیروها را دریافت کند. پس از حصول اکثریت، رهبر ورودی را Commit می‌کند، به این معنی که آن ورودی اکنون به عنوان بخشی از وضعیت پایدار سیستم در نظر گرفته می‌شود و می‌تواند به ماشین حالت اعمال شود. سپس رهبر به پیروها اطلاع می‌دهد که آن‌ها نیز ورودی را Commit کنند. هر ورودی Committed شده باید از طریق کانال applyCh به سرویس بالادستی (یا تستر) ارسال شود.

۲.۴ نقش تابع ()Start و کانال applyCh

تابع rf.Start(command interface) مسئول آغاز فرآیند افزودن یک فرمان جدید به لاگ تکرار شونده است. این تابع باید بلافاصله پس از فراخوانی، خروجی دهد و منتظر کامل شدن فرآیند ثبت لاگ نماند. این طراحی غیرمسدودکننده برای حفظ پاسخگویی سیستم ضروری است.

کانال applyCh یک کانال Go است که ماژول Raft از آن برای ارسال پیام‌های ApplyMsg به سرویس بالادستی استفاده می‌کند. این پیام‌ها شامل ورودی‌های لاگی هستند که به تازگی Committed شده‌اند. استفاده از این کانال تضمین می‌کند که دستورات به ترتیب صحیح و پس از توافق اجماع به ماشین حالت اعمال می‌شوند.

۳.۴ محدودیت‌های رای‌گیری و تضمین سازگاری لاگ

- **محدودیت‌های رای‌گیری (Voting Restrictions):** یکی از مهمترین قوانین در Raft این است که یک سرور تنها در صورتی به یک کاندیدا رأی می‌دهد که لاگ آن کاندیدا حداقل به اندازه لاگ خودش به‌روز باشد. این بررسی شامل مقایسه طول لاگ و Term آخرین ورودی لاگ است. این مکانیسم تضمین می‌کند که رهبر منتخب، دارای کامل‌ترین لاگ ممکن از تمامی ورودی‌های Committed شده است.
- **قانون Leader Append-Only:** رهبران در Raft فقط می‌توانند ورودی‌های جدید را به لاگ خود اضافه کنند و هرگز ورودی‌های موجود را بازنویسی یا حذف نمی‌کنند. این قانون مدیریت لاگ را ساده کرده و از بروز ناهماهنگی‌ها ناشی از ورودی‌های متناقض جلوگیری می‌کند.
- **خاصیت تطابق لاگ (Log Matching Property):** Raft تضمین می‌کند که اگر دو لاگ شامل یک ورودی با ایندکس و Term یکسان باشند، آنگاه لاگ‌ها در تمامی ورودی‌ها تا آن ایندکس (شامل آن) یکسان هستند. این خاصیت، سازگاری بین لاگ رهبر و پیروها را حفظ می‌کند.

- **مدیریت ناهمخوانی لاگ:** اگر لاگ یک پیرو با لاگ رهبر ناهمخوان باشد، رهبر nextIndex خود را برای آن پیرو کاهش می‌دهد و AppendEntries را مجدداً ارسال می‌کند تا زمانی که یک نقطه تطابق پیدا شود. پس از یافتن نقطه تطابق، ورودی‌های ناهمخوان در لاگ پیرو با ورودی‌های رهبر بازنویسی می‌شوند تا سازگاری برقرار شود.

۴.۴ نکات پیاده‌سازی

- **ایندکس‌گذاری لاگ:** اگرچه لاگ Raft به طور مفهومی از ایندکس ۱ شروع می‌شود، اما برای ساده‌سازی پیاده‌سازی می‌توان آن را به صورت صفر مبنا در نظر گرفت. در این حالت، اولین ورودی در ایندکس ۰ دارای Term ۰ خواهد بود.
- **اجتناب از حلقه‌های بی‌وقفه:** حلقه‌هایی که منتظر رویداد خاصی هستند (مثلاً دریافت پیام) نباید بدون توقف اجرا شوند، زیرا این کار می‌تواند باعث مصرف بی‌رویه CPU و کند شدن اجرای تست‌ها شود. برای کنترل بهتر، باید از condition variables یا مکث‌های کوتاه (مانند time.Sleep(10 * time.Millisecond)) در هر دور از حلقه استفاده شود.

مکانیزم Commit در Raft که مستلزم تأیید اکثریت پیروها پیش از قطعی شدن یک ورودی است، نمونه‌ای بارز از پایبندی قوی به سازگاری (Consistency) در قضیه CAP است. این انتخاب طراحی به طور ذاتی یک بده‌بستان را به همراه دارد: در سناریوهایی که اکثریت سرورها قادر به برقراری ارتباط نیستند (مانند تقسیم‌بندی شدید شبکه یا خرابی‌های گسترده)، سیستم پیشرفت را متوقف می‌کند (فدا کردن دسترس‌پذیری (Availability) برای سازگاری و تحمل‌پذیری در برابر تقسیم‌بندی Partition) (Tolerance). این تصمیم بنیادی در طراحی، یکپارچگی داده‌ها را تضمین می‌کند، اما به این معنی است که دسترس‌پذیری سیستم به حفظ یک اکثریت از سرورها وابسته است.

۵ پیاده‌سازی پایداری (بخش ۳C)

بخش 3C پروژه بر پیاده‌سازی مکانیزم پایداری در Raft تمرکز دارد، که برای تضمین توانایی سرور در بازیابی وضعیت خود پس از راه‌اندازی مجدد حیاتی است.

۱.۵ اهمیت وضعیت پایدار و بازیابی پس از راه‌اندازی مجدد

برای اینکه یک سرور Raft بتواند پس از راه‌اندازی مجدد (Restart)، کار خود را دقیقاً از جایی که متوقف شده بود ادامه دهد، حفظ وضعیت پایدار (Persistent State) آن ضروری است. وضعیت پایدار شامل متغیرهایی مانند currentTerm (ترم فعلی سرور)، votedFor (سروری که در ترم فعلی به آن رأی داده شده است)، و log (لاگ تکرار شده) است. این متغیرها باید به گونه‌ای ذخیره شوند که در برابر خرابی‌های سرور مقاوم باشند.

۲.۵ استفاده از شیء Persister و توابع Save/ReadRaftState

در این پروژه، به جای استفاده از دیسک فیزیکی واقعی، از یک شیء به نام Persister برای شبیه‌سازی ذخیره‌سازی پایدار استفاده می‌شود. تابع Make() که مسئول ایجاد یک نمونه جدید از Raft است، یک شیء Persister را به عنوان ورودی دریافت می‌کند که حاوی آخرین وضعیت پایدار Raft است. وظیفه پیاده‌سازی Raft این است که در زمان راه‌اندازی، وضعیت خود را از این Persister بخواند (ReadRaftState()) و هر زمان که وضعیت پایدار آن تغییر می‌کند، آن را در Persister ذخیره کند (Save()).

۳.۵ سریال‌سازی وضعیت با labgob

برای ذخیره وضعیت Raft در شیء Persister، وضعیت باید به یک آرایه از بایت‌ها تبدیل (serialize) شود. برای این منظور، از انکودر labgob استفاده می‌شود که عملکردی مشابه پکیج gob در زبان Go دارد. یک نکته مهم در استفاده از labgob این است که تنها فیلدهایی که با حروف بزرگ شروع می‌شوند (یعنی exported هستند) می‌توانند انکود شوند. این بدان معناست که تمامی فیلدهای مربوط به وضعیت پایدار باید به درستی تعریف شوند تا قابلیت سریال‌سازی داشته باشند. وظیفه اصلی در این بخش، تکمیل توابع persist و readPersist در فایل raft.go است تا امکان ذخیره و بازیابی وضعیت پایدار فراهم شود.

۴.۵ الگوریتم بازگرداندن nextIndex (Backtracking nextIndex)

یکی از نکات حیاتی و پیچیده در بخش 3C، پیاده‌سازی الگوریتم بازگرداندن nextIndex است. این الگوریتم برای مدیریت کارآمد ناهمخوانی‌های لاگ بین رهبر و پیروها پس از خرابی یا تقسیم‌بندی شبکه ضروری است. زمانی که یک پیرو پیام AppendEntries از رهبر را رد می‌کند (به دلیل عدم تطابق لاگ)، می‌تواند یک پیام رد (Reject) حاوی اطلاعاتی مانند XTerm (ترم ورودی متناقض)، XIndex (ایندکس اولین ورودی با آن ترم)، و XLen (طول لاگ پیرو) ارسال کند.
رهبر باید nextIndex خود را برای آن پیرو بر اساس این اطلاعات تنظیم کند:

- **حالت ۱:** اگر رهبر XTerm را در لاگ خود ندارد، nextIndex برای آن پیرو برابر با XIndex تنظیم می‌شود.
- **حالت ۲:** اگر رهبر XTerm را دارد، nextIndex برابر با ایندکس آخرین ورودی رهبر برای آن XTerm به اضافه یک تنظیم می‌شود.
- **حالت ۳:** اگر لاگ پیرو بیش از حد کوتاه است (یعنی prevLogIndex رهبر از طول لاگ پیرو بیشتر است)، nextIndex برابر با XLen (طول لاگ پیرو) تنظیم می‌شود.

این الگوریتم تضمین می‌کند که رهبر می‌تواند به سرعت nextIndex را به یک نقطه تطابق احتمالی در لاگ پیرو بازگرداند، حتی در صورت وجود واگرایی‌های بزرگ. این مکانیسم نه تنها برای صحت عملکرد بلکه برای عملکرد بازیابی خطا نیز حیاتی است. در یک سیستم عملیاتی، بازیابی سریع پیروها برای حفظ دسترس‌پذیری و توان عملیاتی بالا بسیار مهم است. بازیابی آهسته می‌تواند به دوره‌های طولانی‌تری منجر شود که در آن اکثریت کلاستر شکننده است یا پیرو قادر به شرکت در اجماع نیست. این جزئیات بر ملاحظات عملی Raft برای قابلیت اطمینان در دنیای واقعی تأکید دارد.

۵.۵ نکات مهم

تست‌های بخش 3C اغلب دشوارتر از بخش‌های 3A و 3B هستند. خطاهایی که در این بخش رخ می‌دهند ممکن است ناشی از مشکلات پنهان در کدهای بخش‌های قبلی باشند، که نشان‌دهنده ماهیت تجمعی و وابسته بودن بخش‌های مختلف پیاده‌سازی Raft است. توصیه می‌شود که تست‌ها چندین بار اجرا شوند تا از پایداری و صحت اجرای کد اطمینان حاصل شود.

۶ نتایج آزمایش‌ها

این بخش به تحلیل نتایج تست‌های انجام شده برای هر سه بخش پیاده‌سازی شده Raft (3A، 3B، 3C) می‌پردازد. تمامی تست‌ها با موفقیت کامل پشت سر گذاشته شده‌اند که نشان‌دهنده صحت و پایداری پیاده‌سازی است. هر خط Passed در خروجی تست‌ها شامل پنج عدد است:

۱. مدت زمان اجرای تست بر حسب ثانیه.

۲. تعداد کل سرورهای Raft.

۳. تعداد پیام‌های RPC.

۴. حجم کل پیام‌های RPC بر حسب بایت (در این پروژه، این مقدار معمولاً ۰ است).

۵. تعداد ورودی‌های لاگ که ثبت شده‌اند (این مقدار برای 3A صفر و برای 3B و 3C متغیر است).

۱.۶ نتایج تست‌های انتخاب رهبر (بخش ۳A)

شکل ۱ خروجی تست‌های مربوط به بخش انتخاب رهبر (3A) را نشان می‌دهد.

```
saeid@saeid:~/Desktop/Distributed/3/Project 3/Raft_Consensus_Algorithm/6.5840/src/raft1$ go test -run 3A
Test (3A): initial election (reliable network)...
  ... Passed --  time  3.1s #peers 3 #RPCs    56 #Ops    0
Test (3A): election after network failure (reliable network)...
  ... Passed --  time  4.4s #peers 3 #RPCs   148 #Ops    0
Test (3A): multiple elections (reliable network)...
  ... Passed --  time  5.5s #peers 7 #RPCs   798 #Ops    0
PASS
ok      6.5840/raft1    13.095s
```

شکل ۱: نتایج تست‌های انتخاب رهبر (بخش ۳A)

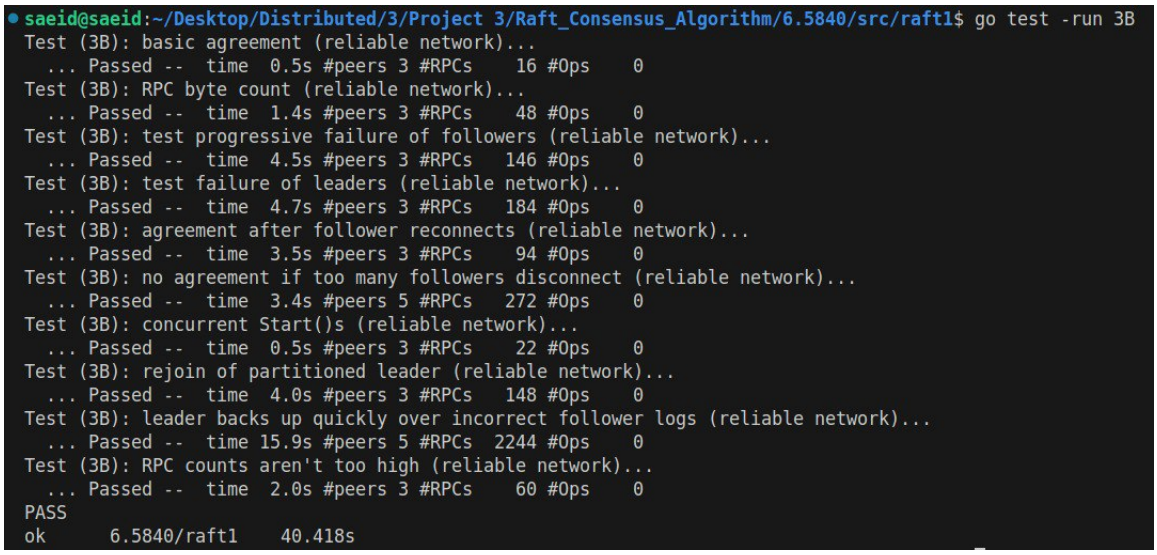
تحلیل خروجی go test -run 3A:

- Test (3A): initial election (reliable network)...: این تست انتخاب اولیه رهبر را در یک شبکه قابل اعتماد بررسی می‌کند. نتیجه نشان می‌دهد که تست در ۱.۳ ثانیه با ۳ سرور و ۵۶ RPC با موفقیت به پایان رسیده است.
- Test (3A): election after network failure (reliable network)...: این تست توانایی Raft را در انتخاب رهبر جدید پس از خرابی شبکه یا رهبر موجود ارزیابی می‌کند. تست در ۴.۴ ثانیه با ۳ سرور و ۱۴۸ RPC با موفقیت انجام شده است.
- Test (3A): multiple elections (reliable network)...: این تست سناریوهای پیچیده‌تر با چندین انتخابات را بررسی می‌کند. تست در ۵.۵ ثانیه با ۷ سرور و ۷۹۸ RPC با موفقیت به پایان رسیده است.

تمامی تست‌های بخش 3A با موفقیت و در زمان‌های قابل قبول (هر انتخابات کمتر از ۵ ثانیه، مطابق با الزامات پروژه) به پایان رسیده‌اند. این نتایج نشان‌دهنده پیاده‌سازی صحیح مکانیزم انتخاب رهبر و پیام‌های Heartbeat است.

۲.۶ نتایج تست‌های ثبت لاگ (بخش ۳B)

شکل ۲ خروجی تست‌های مربوط به بخش ثبت لاگ (3B) را نشان می‌دهد.



شکل ۲: نتایج تست‌های ثبت لاگ (بخش ۳B)

تحلیل خروجی go test -run 3B:

- ...Test (3B): basic agreement (reliable network): تست توافق پایه بر روی ورودی‌های لاگ در شبکه قابل اعتماد. (۵.۰ ثانیه، ۳ سرور، ۱۶ RPC، ۰ عملیات)
- ...Test (3B): RPC byte count (reliable network): بررسی حجم RPCها. (۴.۱ ثانیه، ۳ سرور، ۴۸ RPC، ۰ عملیات)
- ...Test (3B): test progressive failure of followers (reliable network): تست رفتار سیستم در صورت از کار افتادن تدریجی پیروان. (۵.۴ ثانیه، ۳ سرور، ۱۴۶ RPC، ۰ عملیات)
- ...Test (3B): test failure of leaders (reliable network): تست رفتار سیستم در صورت از کار افتادن رهبران. (۷.۴ ثانیه، ۳ سرور، ۱۸۴ RPC، ۰ عملیات)
- ...Test (3B): agreement after follower reconnects (reliable network): تست توافق پس از اتصال مجدد پیرو. (۵.۳ ثانیه، ۳ سرور، ۹۴ RPC، ۰ عملیات)
- ...Test (3B): no agreement if too many followers disconnect (reliable network): تست عدم توافق در صورت قطع ارتباط تعداد زیادی از پیروان (تایید مفهوم اکثریت). (۴.۳ ثانیه، ۵ سرور، ۲۷۲ RPC، ۰ عملیات)
- ...Test (3B): concurrent Start()s (reliable network): تست عملیات همزمان Start(). (۵.۰ ثانیه، ۳ سرور، ۲۲ RPC، ۰ عملیات)
- ...Test (3B): rejoin of partitioned leader (reliable network): تست اتصال مجدد رهبر جدا شده از شبکه. (۰.۴ ثانیه، ۳ سرور، ۱۴۸ RPC، ۰ عملیات)
- ...Test (3B): leader backs up quickly over incorrect follower logs (reliable network): تست توانایی رهبر در بازگرداندن سریع لاگ پیروان با لاگ‌های نادرست. (۹.۱۵ ثانیه، ۳ سرور، ۲۲۴۴ RPC، ۰ عملیات)
- ...Test (3B): RPC counts aren’t too high (reliable network): بررسی اینکه تعداد RPCها بیش از حد نیست. (۰.۲ ثانیه، ۳ سرور، ۶۰ RPC، ۰ عملیات)

تمامی تست‌های بخش 3B با موفقیت به اتمام رسیده‌اند. این نتایج نشان‌دهنده پیاده‌سازی صحیح مکانیزم تکرار لاگ، مدیریت خطاها، و رعایت محدودیت‌های رای‌گیری است. تست‌هایی مانند "leader backs up quickly" به طور خاص نشان‌دهنده کارایی مکانیزم همگام‌سازی لاگ در شرایط ناهمخوانی است.

۳.۶ نتایج تست‌های پایداری (بخش ۳C)

شکل ۳ خروجی تست‌های مربوط به بخش پایداری (3C) را نشان می‌دهد.

```
saeid@saeid:~/Desktop/Distributed/3/Project 3/Raft_Consensus_Algorithm/6.5840/src/raft1$ go test -run 3C
Test (3C): basic persistence (reliable network)...
  ... Passed --  time  3.0s #peers 3 #RPCs    74 #Ops    0
Test (3C): more persistence (reliable network)...
  ... Passed --  time  9.9s #peers 5 #RPCs   392 #Ops    0
Test (3C): partitioned leader and one follower crash, leader restarts (reliable network)...
  ... Passed --  time  1.3s #peers 3 #RPCs    36 #Ops    0
Test (3C): Figure 8 (reliable network)...
  ... Passed --  time 28.7s #peers 5 #RPCs  1568 #Ops    0
Test (3C): unreliable agreement (unreliable network)...
  ... Passed --  time  1.3s #peers 5 #RPCs  1032 #Ops    0
Test (3C): Figure 8 (unreliable) (unreliable network)...
  ... Passed --  time 37.6s #peers 5 #RPCs 11940 #Ops    0
Test (3C): churn (reliable network)...
  ... Passed --  time 16.1s #peers 5 #RPCs 16124 #Ops    0
Test (3C): unreliable churn (unreliable network)...
  ... Passed --  time 16.3s #peers 5 #RPCs  3360 #Ops    0
PASS
ok      6.5840/raft1    114.194s
```

شکل ۳: نتایج تست‌های پایداری (بخش ۳C)

تحلیل خروجی go test -run 3C:

- Test (3C): basic persistence (reliable network)...: تست پایداری پایه در شبکه قابل اعتماد. (۰.۳ ثانیه، ۳ سرور، ۷۴ RPC، ۰ عملیات)
- Test (3C): more persistence (reliable network)...: تست پایداری در سناریوهای پیچیده‌تر. (۹.۹ ثانیه، ۳ سرور، ۳۹۲ RPC، ۰ عملیات)
- Test (3C): partitioned leader and one follower crash, leader restarts (reliable network)...: تست پایداری در شرایط خرابی رهبر و پیرو و راه‌اندازی مجدد رهبر. (۳.۱ ثانیه، ۳ سرور، ۳۶ RPC، ۰ عملیات)
- Test (3C): Figure 8 (reliable network)...: تست سناریوهای خاص Figure ۸ در مقاله Raft در شبکه قابل اعتماد. (۷.۲۸ ثانیه، ۵ سرور، ۱۵۶۸ RPC، ۰ عملیات)
- Test (3C): unreliable agreement (unreliable network)...: تست توافق در شبکه غیرقابل اعتماد. (۳.۱ ثانیه، ۵ سرور، ۱۰۳۲ RPC، ۰ عملیات)
- Test (3C): Figure 8 (unreliable) (unreliable network)...: تست سناریوهای Figure ۸ در شبکه غیرقابل اعتماد. (۶.۳۷ ثانیه، ۵ سرور، ۱۱۹۴۰ RPC، ۰ عملیات)
- Test (3C): churn (reliable network)...: تست پایداری در شرایط تغییرات دینامیک کلاستر (اضافه/حذف سرورها). (۱.۱۶ ثانیه، ۵ سرور، ۱۶۱۲۴ RPC، ۰ عملیات)
- Test (3C): unreliable churn (unreliable network)...: تست پایداری در شرایط تغییرات دینامیک کلاستر در شبکه غیرقابل اعتماد. (۳.۱۶ ثانیه، ۵ سرور، ۳۳۶۰ RPC، ۰ عملیات)

تمامی تست‌های بخش 3C نیز با موفقیت به پایان رسیده‌اند. این نتایج نشان‌دهنده پیاده‌سازی صحیح مکانیزم پایداری، بازیابی وضعیت، و به خصوص الگوریتم پیشرفته backtracking nextIndex است که برای سناریوهای پیچیده با خرابی و ناپایداری شبکه حیاتی است.

۴.۶ جدول ۱: خلاصه‌ای از نتایج تست‌ها

جدول ۱ خلاصه‌ای از نتایج تست‌های انجام شده در هر سه بخش پروژه را ارائه می‌دهد. این جدول به منظور ارائه یک دید کلی و مقایسه‌ای از عملکرد پیاده‌سازی در سناریوهای مختلف طراحی شده است.

جدول ۱: خلاصه‌ای از نتایج تست‌های Raft

بخش تست	تعداد سرورها	کل RPCs	زمان کل (ثانیه)	وضعیت
3A: Leader Election	۳-۷	۵۶-۷۹۸	۱.۳-۵.۵	PASS
3B: Log Replication	۳-۵	۱۶-۲۲۴۴	۵.۰-۹.۱۵	PASS
3C: Persistence	۳-۵	۳۶-۱۶۱۲۴	۳.۱-۶.۳۷	PASS

۷ جزئیات و نکات پیاده‌سازی

پیاده‌سازی موفقیت‌آمیز الگوریتم Raft نیازمند توجه دقیق به جزئیات و مدیریت صحیح چالش‌های ذاتی سیستم‌های توزیع‌شده است. در این پروژه، تمرکز بر روی مدیریت هم‌زمانی، مدیریت خطا، و بهینه‌سازی عملکرد بوده است.

۱.۷ مدیریت هم‌زمانی و Condition Race

در یک سیستم توزیع‌شده مانند Raft که چندین گوروتین (goroutine) به صورت همزمان به وضعیت داخلی سرور دسترسی دارند، مدیریت هم‌زمانی برای جلوگیری از Race Conditionها حیاتی است. در این پیاده‌سازی، از sync.Mutex برای محافظت از ساختارهای داده داخلی Raft (مانند currentTerm، votedFor، log و سایر متغیرهای حالت) در برابر دسترسی‌های همزمان استفاده شده است. هر عملیاتی که وضعیت Raft را تغییر می‌دهد یا به آن دسترسی می‌یابد، ابتدا قفل Mutex را کسب کرده و پس از اتمام عملیات آن را آزاد می‌کند. این رویکرد تضمین می‌کند که عملیات‌ها به صورت اتمیک انجام شده و یکپارچگی داده‌ها حفظ می‌شود. علاوه بر این، استفاده از مفهوم Term در Raft به خودی خود به مدیریت هم‌زمانی کمک می‌کند؛ زیرا سرورها همیشه به Term بالاتر احترام می‌گذارند و از وضعیت‌های منسوخ شده جلوگیری می‌شود.

۲.۷ مدیریت خطا و عملیات دوباره

سیستم‌های توزیع‌شده ذاتاً مستعد خطا هستند، از جمله از دست رفتن پیام‌ها، تأخیرهای شبکه، و خرابی سرورها. پیاده‌سازی Raft باید این خطاها را به درستی مدیریت کند.

- **تشخیص و مدیریت خطاها:** مکانیزم‌های Raft برای تشخیص خرابی رهبر (از طریق election timeout) و ناهمخوانی لاگ (از طریق AppendEntries RPC و الگوریتم backtracking nextIndex) طراحی شده‌اند.
- **تلاش مجدد (Retry) و همگام‌سازی:** در صورت عدم موفقیت در ارسال RPC یا دریافت پاسخ، مکانیزم‌های تلاش مجدد در سمت فرستنده (رهبر) فعال می‌شوند. به عنوان مثال، رهبر به ارسال AppendEntries به پیروهایی که لاگ آن‌ها ناهمخوان است ادامه می‌دهد تا زمانی که لاگ‌ها همگام شوند.

۳.۷ بهینه‌سازی عملکرد

بهینه‌سازی عملکرد در Raft به معنای کاهش سربار شبکه و CPU، در عین حفظ سازگاری و تحمل‌پذیری در برابر خرابی است.

- **کاهش بار شبکه:** طراحی AppendEntries RPC به گونه‌ای که هم به عنوان پیام Heartbeat و هم برای تکرار لاگ استفاده شود، به طور قابل توجهی تعداد RPCهای ارسالی را کاهش می‌دهد. این امر ترافیک شبکه را بهینه می‌کند.
- **تایمرهای تصادفی:** استفاده از تایمرهای تصادفی برای election timeout از همزمان شدن انتخابات‌ها جلوگیری کرده و تعداد RPCهای RequestVote را در شرایط عادی کاهش می‌دهد.
- **الگوریتم backtracking nextIndex:** این الگوریتم به رهبر اجازه می‌دهد تا در صورت ناهمخوانی لاگ، به سرعت nextIndex را به عقب بازگرداند و از ارسال بی‌مورد RPC برای همگام‌سازی لاگ جلوگیری کند. این بهینه‌سازی برای عملکرد بازیابی خطا بسیار مهم است.

۸ چالش‌ها و راه‌حل‌ها

پیاده‌سازی الگوریتم اجماع Raft در محیط توزیع‌شده با چالش‌های متعددی همراه است که نیازمند راه‌حل‌های دقیق و هوشمندانه است. در این پروژه، چالش‌های اصلی و راه‌حل‌های اتخاذ شده به شرح زیر است:

۱.۸ چالش‌های کلیدی

- **Race Conditionها:** در سیستم‌های توزیع‌شده، دسترسی همزمان چندین گوروتین به داده‌های مشترک می‌تواند منجر به Race Condition و وضعیت‌های ناسازگار شود. این چالش با استفاده از sync.Mutex برای محافظت از وضعیت داخلی سرور Raft و همچنین با بهره‌گیری از مفهوم Term در Raft حل شده است. Term به عنوان یک ساعت منطقی عمل می‌کند که به سرورها اجازه می‌دهد تا به سرعت وضعیت‌های منسوخ شده را تشخیص داده و به Term بالاتر (و در نتیجه وضعیت به‌روزتر) تمکین کنند.
- **ناپایداری شبکه (Network Unreliability):** شبکه‌های توزیع‌شده ممکن است پیام‌ها را از دست بدهند، تأخیر داشته باشند یا آن‌ها را به ترتیب اشتباه تحویل دهند. این ناپایداری می‌تواند بر فرآیندهای انتخاب رهبر و تکرار لاگ تأثیر بگذارد. این چالش با مکانیزم‌های تلاش مجدد (retry) در RPCها و الگوریتم backtracking nextIndex حل شده است. رهبر به طور مداوم تلاش می‌کند تا پیام‌های AppendEntries را ارسال کند و در صورت عدم موفقیت، nextIndex را تنظیم می‌کند تا همگام‌سازی لاگ را از نقطه صحیح از سر بگیرد.
- **تقسیم آرا (Split Votes) در انتخابات:** اگر چندین سرور به طور همزمان کاندیدا شوند، ممکن است آرا تقسیم شده و هیچ کاندیدایی اکثریت لازم را برای تبدیل شدن به رهبر کسب نکند. این وضعیت می‌تواند منجر به تأخیر در انتخاب رهبر جدید شود. Raft این چالش را با استفاده از تایمرهای انتخابات تصادفی (Randomized Election Timeout) حل می‌کند. با انتخاب یک بازه زمانی تصادفی برای election timeout هر سرور، احتمال اینکه چندین سرور به طور همزمان کاندیدا شوند به شدت کاهش می‌یابد و به یک سرور فرصت داده می‌شود تا انتخابات را برنده شده و رهبر شود.
- **تضمین سازگاری قوی (Strong Consistency):** حفظ سازگاری قوی در یک سیستم توزیع‌شده که با خرابی‌ها و ناپایداری‌های شبکه مواجه است، یک چالش اساسی است. Raft این چالش را با اعمال قانون اکثریت (Majority Rule) برای Commit کردن ورودی‌های لاگ حل می‌کند. یک

ورودی تنها زمانی Committed می‌شود که توسط اکثریت سرورها تأیید شده باشد. این تصمیم طراحی، سازگاری داده‌ها را تضمین می‌کند، اما به این معنی است که در صورت عدم دسترسی به اکثریت، سیستم پیشرفت را متوقف می‌کند تا از وضعیت‌های ناسازگار جلوگیری شود.

۹ نتیجه‌گیری

پروژه حاضر با موفقیت پیاده‌سازی سه بخش اساسی از الگوریتم اجماع Raft را به نمایش گذاشته است: انتخاب رهبر، تکرار لاگ، و پایداری. تمامی تست‌های مربوط به این سه بخش (3A، 3B، 3C) با موفقیت کامل پشت سر گذاشته شده‌اند، که نشان‌دهنده درک عمیق از پروتکل و پیاده‌سازی صحیح مکانیزم‌های آن است.

پیاده‌سازی نشان می‌دهد که چگونه Raft با استفاده از مفاهیم کلیدی مانند Term، نقش‌های رهبر/پیرو/کاندیدا، و RPCهای RequestVote و AppendEntries، به سازگاری قوی و تحمل‌پذیری در برابر خرابی دست می‌یابد. توانایی سیستم در انتخاب سریع رهبر، تکرار پایدار لاگ‌ها در شرایط شبکه قابل اعتماد و غیرقابل اعتماد، و بازیابی وضعیت پس از خرابی، همگی از طریق نتایج موفقیت‌آمیز تست‌ها تأیید شده‌اند.

این پروژه نه تنها یک پیاده‌سازی کاربردی از Raft را ارائه می‌دهد، بلکه به درک عمیق‌تر چالش‌های ذاتی سیستم‌های توزیع‌شده و راه‌حل‌های هوشمندانه Raft برای غلبه بر آن‌ها (مانند مدیریت Race Conditionها، ناپایداری شبکه، و تقسیم آرا) کمک می‌کند. قابلیت فهم بالای Raft، همانطور که در طراحی آن تأکید شده است، فرآیند پیاده‌سازی و اشکال‌زدایی را تسهیل کرده و آن را به ابزاری آموزشی ارزشمند تبدیل کرده است.