سیستم‌های توزیع شده

تمرین سری چهارم

استاد: دکتر کمندی

پارسا محمدپور

پاییز ۱۴۰۳

پرسش یک

با تغییر و اصلاح الگوریتم Two-Phase Commit، الگوریتم جدیدی پیشنهاد دهید که وابستگی به یک coordinator واحد را کاهش دهد. نحوه عملکرد الگوریتم خود را به صورت کامل شرح دهید. تاثیر این تغییر را بر پیچیدگی الگوریتم ارزیابی کنید.

برای حل این سوال ابتدا توضیح مختصری در رابطه با الگوریتم Two-Phase Commit ارائه می‌دهیم.

الگوریتم Two-Phase Commit:

این الگوریتم به این صورت است که فرآیند تصمیم‌گیری پراسس‌ها در حداکثر دو راند صورت می‌گیرد. روال کلی این الگوریتم در دو راند اجرا می‌شود:

در راند اول، تمامی پراسس‌هابه جز پراسسی که به عنوان Coordinator انتخاب شده‌است، مقدار اولیه خودشان را برای پراسس Coordinator ارسال می‌کنند و هر پراسسی که مقدار اولیه آن برابر با یک است، بر روی مقدار یک تصمیم‌گیری می‌کند. پراسسی که به عنوان Coordinator انتخاب شده‌است، این مجموعه مقادیر دریافتی را جمع می‌کند (و در یک بردار می‌ریزد) و اگر تمامی آن مقادیر دریافتی برابر با یک باشد، آن مقدار را به عنوان تصمیم نهایی اتخاذ می‌کند و سپس آن را به همه اعلام می‌کند و همه در این راند تصمیم یک را اتخاذ می‌کنند. اما اگر تمام مقادیر دریافتی برابر با یک نباشند، آن وقت پراسس Coordinator مقدار صفر را به عنوان تصمیم نهایی درنظر می‌گیرد و سپس آن را به بقیه اعلام می‌کند. (حتی اگر پراسسی هم در ارسال پیام در راند اول با شکست مواجه شود، یعنی قبل از ارسال پیام دچار مشکل شود و پیام را ارسال نکند، آن وقت باز هم تصمیم صفر را اتخاذ می‌کنیم.)

در رابطه با پیچیدگی پیامی این الگوریتم هم باید بگوییم که در **دو راند** انجام می‌شود، پس پیچیدگی زمانی آن دو است. همچنین در رابطه با پیچیدگی پیامی هم باید بگوییم که پیچیدگی پیامی آن در حالتی که هیچ خطایی رخ ندهد، برابر با **2(n-1)** می‌باشد.

حالا با توجه به گفته صورت سوال، اگر بخواهیم در این الگوریتم تغییری ایجاد کنیم تا وابستگی آن به یک Coordinator واحد کم شود، می‌توانیم این کار را انجام دهیم:

برای این این مورد می‌توانیم **بیشتر از یک Coordinator** داشته باشیم. برای این منظور، مثلا پراسس‌های یک و دو را به عنوان Coordinator ها انتخاب کنیم. سپس در راند اول تمام پراسس‌ها مقدار اولیه‌شان را برای هر دو پراسس یک و دو ارسال می‌کنند سپس مشابه با الگوریتم عادی، اگر مقدارشان صفر است، تصمیم صفر را اتخاذ می‌کنند. سپس هر دو پراسس یک و دو، مشابه الگوریتم عادی تصمیم نهایی را مشخص می‌کنند. (اگر از همه پراسس‌ها مقدار یک را دریافت کرده باشن تصمیم یک را می‌گیرند؛ در غیراینصورت (چه مقدار صفر دریافت کرده باشند چه دراصلا هیچ مقداری دریافت نکرده باشند) تصمیم صفر را اتخاذ می‌کنند) سپس این تصمیم را به تمام پراسس‌ها انتقال می‌دهند. سپس، پس از پایان انتقال پیام‌ها، اگر همه پراسس‌ها پیام‌های دریافتی از هر دو پراسس یک و دو یکسانی داشتند (و هنوز تصمیم نگرفته بودند) روی آن مقدار تصمیم می‌گیرند؛ در غیراینصورت بر روی مقدار صفر تصمیم می‌گیرند.

در رابطه با پیچیدگی این الگریتم باید گفت که مشابه با الگوریتم عادی، پیچیدگی زمانی آن همان **دو راند** باقی می‌ماند ولی از لحاظ پیچیدگی پیامی، تعداد پیام‌ها در حالتی که خطا نداشته باشیم (بیشترین مقدار تبادیلپیام وجود داشته باشد) **دو برابر** حالت قبل می‌شود. یعنی پیچیدگی پیامی برابر با 4(n-1) می‌شود.

پرسش دو

یک تغییر برای الگوریتم Three-Phase Commit پیشنهاد دهید که در شرایط بدون شکست (Failure-Free)، فرآیندها بتوانند سریع تصمیم‌گیری کرده و متوقف شوند. الگوریتم شما باید دارای تعداد کمی دور و تعداد پیام از مرتبه O(n) باشد. نحوه عملکرد الگوریتم خود را به صورت کامل شرح دهید. درستی این الگوریتم را اثبات کنید.

برای این کار می‌توانیم به اصلاحاتی به سراغ الگوریتم Two-Phase Commit برویم. به وطریکه، هر پراسسی که مقدار اولیه آن برابر با صفر است، بر روی صفر تصمیم بگیرد و آن مقدار را برای پراسس Coordinator ارسال کند. سپس Coordinator به محض دریافت پیام صفر، بر روی صفر تصمیم‌گیری کند و آن را برای همه ارسال کند و بقیه به محض دریافت این پیام بر روی مقدار صفر تصمیم‌گیری کنند.(اگر پراسسی که مقدار اولیه صفر را داشت همان Coordinator بود، باز هم تصمیم صفر را می‌گیرد و آن را برای همه ارسال می‌کند و در همان راند اول همه تصمیم صفر را اتخاذ می‌کنند.) در غیر اینصورت، اگر پراسس Coordinator مقدار یک را از همه دریافت کرده بود (و همچنین مقدار خودش هم برابر با یک بود)، در راند بعدی آن مقدار را برای همه ارسال می‌کند و بر روی آن مقدار یک تصمیم می‌گیرد. در رابطه با پیچیدگی زمانی، باید بگوییم این الگوریتم در دو راند اجرا می‌شود. از لحاظ پیچیدگی پیامی هم، تعداد پیام‌های ردوبدل شده، از اردر O(n) می‌باشد. چون در هر راند حداکثر n تا پیام ارسال می‌شود. (در حالتی که پراسسی مقدار اولیه آن صفر باشد، پیام صفر را می‌فرستد و بر روی آن تصمیم‌گیری می‌کند و پراسس Coordinator نیازی ندارد تا تصمیم نهایی که صفر است را دوباره برای آن ارسال کند. برای همین می‌شود این الگوریتم را طوری پیاده‌سازی کرد که کمتر از این تعداد پیام در یک سری حالات قابل انجام باشد، ولی حداکثر این تعداد پیام نیاز است.)

برای اثبات درستی این الگوریتم هم باید اثبات کنیم که هر سه مورد Agreement، Validity و Weak Termination رعایت می‌شوند. پس به ترتیب آن‌ها را اثبات می‌کنیم:

- **Agreement**:

برای این مورد مشخص است. فرض می‌کنیم که یک پراسس تصمیم یک را گرفته و دیگری تصمیم صفر را گرفته است سپس با توجه به اینکه در انتها به تناقض می‌رسیم، اثبات می‌شود که همچین چیزی امکان ندارد و Agreement برقرار است. برای اینکه این پراسس تصمیم یک را بگیرد، باید از Coordinator مقدار یک را به عنوان تصمیم یک دریافت کرده باشد و همچنین پراسس دیگر نیز برای اینکه تصمیم صفر را اتخاذ کند، یا باید مقدار اولیه خودش برابر با صفر بوده باشد یا مقدار صفر را از Coordinator شنیده باشد. که در هر دو صورت هم به تناقض می‌خوریم. چون اگر مقدار خودش برابر با صفر بوده باشد و تصمیم صفر را گرفته باشد، پس آن مقدار صفر را هم برای Coordinator ارسال کرده است و Coordinator چون پیام صفر را دریافت کرده است، پس نمی‌تواند تصمیمی جز صفر بگیرد طبق روند تصمیم‌گیری در الگوریتم پیشنهاد شده و نمی‌توانسه به پراسس دیگر که تصمیم یک را گرفته، مقدار یک را گفته باشد. اما اگر هم مقدار صفر را از Coordinator شنیده باشد، باز هم به صورت بدیهی به تناقض می‌خوریم چون Coordinator به یک پراسس مقدار صفر را به عنوان تصمیم نهایی گفته است و به دیگری مقدار یک را به عنوان تصمیم نهایی گفته است. که باز هم به تناقض رسیدیم. چون اگر Coordinator مقداری برابر با صفر را دریافت کرده باشد و یا مقدار خودش برابر با صفر باشد تصمیم صفر را می‌گیرد و دیگر به کسی تصمیم یک را نمی‌گوید و اگر هم تصمیم یک را گرفته باشد، به این معنی است که از همه پراسس‌ها مقدار یک را دریافت کرده است و مقدار خودش هم یک بوده است، پس هیچ‌جایی نمی‌توانسته تصمیم یک را برای پراسسی ارسال کرده باشد. پس به تناقض می‌خوریم و دو پراسس نمی‌توانند دو تصمیم متفاوت بگیرند. پس Agreement برقرار است.

- **Validity**:

برای Validity هم مشخص است، اگر پراسسی تصمیم صفر بگیرد، آن مقدار را برای Coordinator ارسال می‌کند و آن هم به محض دریافت آن مقدار تصمیم صفر را می‌گیرد و برای همه آن را ارسال می‌کنند و همه تصمیم صفر می‌گیرند. مخصوصا چون پراسس‌ها دچار خطا نمی‌شوند، پس مشکلی هم در ارسال و دریافت پیام و خطادارد شدن پراسس نیست. پس همه چیز مطابق انتظار پیش می‌رود. اما اگر مقدار همه پراسس‌ها برابر با یک باشد، هیچکدام از آن‌ها در مرحله یک تصمیم نمی‌گیرند و فقط مقدار پیام خودشان را برای پراسس Coordinator ارسال می‌کنند. در انتهای این راند هم با توجه به اینکه پراسس Coordinator مقدار یک را از همه پراسس‌ها دریافت کرده، تصمیم یک را اتخاذ می‌کند و برای همه آن را ارسال می‌کند و همه تصمیم یک می‌گیرند. پس Validity برقرار است. نکته کلیدی این است که همه پیام‌ها اراسل می‌شوند و هیچ پراسسی دچار خطا نمی‌شود و تا انتها به کار خود ادامه می‌دهد.

- **Termination**:

در این الگوریتم، تحت اگر پراسسی مقدار صفر را به عنوان مقدار اولیه داشته باشد، در همان ابتدا تصمیم می‌گیرد و تصمیم آن صفر است و terminate می‌کند. اما اگر مقدار آن یک باشد، ابتدا تصمیم را برای پراسس Coordinator ارسال می‌کند. سپس Coordinator پس از دریافت تمام پیام‌ها از تمام پراسس‌ها و نحوه تصمیم‌گیری تصمیم می‌گیرد و آن تصمیم را برای همه ارسال می‌کند و خودش هم بعد از ارسال تصمیمی که اتخاذ کرده بود را می‌گیرد و Terminate می‌کند. پراسس‌های دیگر نیز با توجه به دریافت تصمیم Coordinator، تصمیم نهایی‌شان مشخص می‌شود و Terminate می‌کنند. نکته کلیدی باز هم این است که چون نه خطای لینک داریک و نه خطای پراسس، همه پیام‌ها ارسال می‌شود و مشکلی برای پراسسی پیش نمی‌آید که مثلا تصمیم بگیرد ولی نتواند آن را برای کسی ارسال کند و یا ... . و برخلاف الگوریتم Two-Phase Commit، در اینجا چون سیستم بدون خطاست، پس امکان این که پراسس Coordinator تصمیم‌گیری نکند یا پیامی را نتواند برای کسی ارسال کند وجود ندارد، پس هر دو شرط Weak Termination و Strong Termination را داریم.

پرسش سه

الگوریتم FloodSet و Two-Phase Commit را در نظر بگیرید.از ترکیب این دو الگوریتم یک الگوریتم جدید برای تصمیم‌گیری ارائه دهید. توجه داشته باشید که الگوریتم باید شرایط مربوط به اجماع را داشته باشد. نحوه عملکرد الگوریتم خود را به صورت کامل شرح دهید و اثبات کنید که الگوریتم می‌تواند شرایط مربوط به اجماع را برآورده کند.

الگوریتم حاصل به این صورت است که در نظر می‌گیریم تمام هر پراسس یک مقدار اولیه‌ای دارد (که این سری می‌تواند هر مقداری باشد و دیگر مصل مسئله Two-Phase Commit منحصر به صفر و یک نیست) و همچنین یک پراسس یه عنوان Coordinator نیز داریم. در این الگوریتم، پراسس Coordinator یک متغیر به نام W را در نظر می‌گیرد که یک مجموعه است و در ابتدای کار مقدار تصمیم خودش را در آن مجموعه قرار می‌دهد.سپس در راند اول، تمام پراسس‌ها مقدار اولیه‌شان را برای پراسس Coordinator ارسال می‌کنند. سپس این پراسس Coordinator مقدار دریافتی از سمت هر کدام از پراسس‌ها را در مجموعه خودش اضافه می‌کند. در انتهای این راند، وقتی Coordinator پیام تمام پراسس‌ها را پردازش کرد و کارش تمام شد، اگر تعداد اعضای مجموعه یکی بود (یعنی مقدار اولیه تمام پراسس‌ها و همچنین خود Coordinator یکسان بود)، روی آن مقدار تصمیم‌گیری صورت می‌گیرد و Coordinator آن مقدار را در راند بعدی برای همه ارسال می‌کند و در انتها هم خودش بر روی آن تصمیم می‌گیرد. در غیر این صورت (یعنی اگر تعداد اعضای مجموعه W در پراسس Coordinator بیشتر از یکی بود) سپس مطابق با الگوریتم FloodSet، تصمیم default اتخاذ می‌شود و پراسس Coordinator مقدار تصمیم default را برای بقیه ارسال می‌کند و در انتها آن تصمیم را اتخاذ می‌کند.

برای اثبات اینکه این الگوریتم، شرایط مربوط به اجماع را برآورده می‌کند، باید اثبات کنیم که این الگوریتم Agreement،Validity و Termination را دارد. در ادامه هر کدام از ا«ها را یکی یکی اثبات می‌کنیم.

- **Agreement**:

فرض می‌کنیم که دو تا پراسس، بر روی دو مقدار مختلف تصمیم‌گیری کرده باشند. این یعنی اینکه باید پراسس Coordinator برای آن‌ها، دو مقدار متفاوت را ارسال کرده باشد که این ممکن نیست؛ زیرا در الگوریتم تصمیم Coordinator بعد از دریافت تمام پیام‌ها اتخاذ می‌شد و در هیچ جایی هم تغییر نمی‌کرد، بنابراین ممکن نیست که Coordinator دو تا تصمیم متفاوت را برای دو پراسس ارسال کرده باشد. پس به **تناقض** می‌خوریم. پس شرط Agreement باید برقرار باشد.

- **Validity**:

در این الگوریتم، اگر تمام پراسس‌ها با مقدار تصمیم اولیه یکسان شروع به فعالیت کنند، چه دچار خطا بشوند چه دچار خطا نشوند، باز مقدار یکسانی را به Coordinator ارسال می‌کنند؛ یعنی Coordinator در هنگام تصمیم گیری یا پیام هر پراسس را دریافت کرده (که همان مقدار یکسان گفته شده است) یا آن مقدار را دریافت نکرده و آنن پراسس دچار خطا شده است و اصلا مقدارش را برای پراسس Coordinator ارسال نکرده استو پس در هنگام تصمیم‌گیری، تنها مقدار موجود در مجموعه W موجود در Coordinator برابر با همان مقدار اولیه گفته شده است و طبق نحوه تصمیم‌گیری، آن مقدار به عنوان تصمیم اتخاذ می‌شود و به وسیله Coordinator برای همه پراسس‌ها ارسال می‌شود. پس تمام پراسس‌ها این مقدار را دریافت می‌کنند و روی آن تصمیم‌گیری انجام می‌دهند. پس تصمیم همه پراسس‌ها یکسان می‌شود. حالا اما اگر خود Coordinator قبل از ارسال همه یا تعدادی از پیام‌های تصمیم به بقیه پراسس‌ها دچار خطا شد چی؟ در این صورت باز هم شرایط برای Validity برقرار است زیرا در مورد شرط Validity ما این را داریم که "اگر همه پراسس‌ها با مقدار یکاسن کار را شروع کنند، آن مقدار **تنها گزینه ممکن** برای تصمیم نهایی باید باشد" و طبق این تعریف، اگر پراسس‌ها تصمیم‌گیری نکنند مشکلی نیست، ولی اگر تصمیم‌گیری کنند، تنها تصمیم مورد قبول، تصمیم اولیه گفته شده است که در اینجا هم اثبات کردیم اگر تصمیم‌گیری توسط پراسس‌ها صورت بگیرد، بر روی همان مقدار است. پس Validity نیز برقرار است.

- **Termination**:

در این الگوریتم، درصورتی که پراسس Coordinator همان پراسسی باشد که دچار خطا می‌شود، ما به مشکل می‌خوریم و ممکن است که شرط Termination برقرار نشود. درست مانند حالتی که در Two-Phase Commit داشتیم که ممکن بود پراسس Coordinator درست بعد از تصمیم‌گیری و یا کلا قبل از فرستادن پیام برای دیگر پراسس‌ها دچار خطا شود و این ممکن بود باعث این شود که کلا پراسس‌ها تصمیم‌گیری نکنند، اینجا هم دقیقا در همین حالت باعث می‌شود که پراسس‌ها نتوانند تصمیم‌گیری کنند و Termination برای آنها اتفاق نیفتد. اما اگر پراسس Coordinator دچار خطا نشود، باعث می‌شود که ما شرط Termination را داشته باشیم. پراسس‌هایی که خطادار نیستند، حتما در انتهای کار، پیامی را از طرف پراسس Coordinator دریافت می‌کنند (چون در این حالت درنظر گرفتیم که Coordinator دچار خطا نمی‌شود) پس پیامی دریافت می‌کنند و یا دچار خطا می‌شوند، که خب باز برای ما مهم نیستند چون شرط Termination برای پراسس‌های بدون خطا بود، یا دچار خطا نمی‌شوند که خب در این صورت طبق روال الگوریتم، تصمیمی که برای‌شان ارسال‌شده است را به عنوان تصمیم نهایی اتخاذ می‌کنند.