УДК 004.42

ИССЛЕДОВАНИЕ АРХИТЕКТУРЫ БЕЗОПАСНОГО ДОСТУПА К ПОЛЯМ СТРУКТУР ХРАНИЛИЩА И СТАТИЧЕСКОГО АНАЛИЗА ГОНОК ЧЕРЕЗ ЛИНТЕР В ПРОЕКТАХ НА GO

Веселов И. П. (ИТМО)
Научный руководитель – преподаватель практики Петухов П. Н.
(ИТМО)

Введение

В высоконагруженных микросервисах на Go обрабатывается сотни тысяч запросов одновременно. Из-за этого проблемы с race condition возникают очень часто, особенно если доступ осуществляется к общим структурам [1]. Мьютексы, заточенные под частые чтения, вроде sync.RWMutex, или атомики, по идее должны решать данную проблему, но под нагрузкой они могут создавать узкие места. Так, каждый вызов RLock() будет менять внутренний счетчик, что приведет к инвалидации кэш-линии на ядрах процессора. Получается такой парадокс: мы стараемся бороться с гонками данных и используем параллельное чтение, но фактически все равно замедляем нашу программу.

Основная часть

Для повышения производительности предлагается другой архитектурный подход. Защищаемая структура с данными не будет вообще содержать мьютексы или атомики внутри. Вместо этого мы будем делить код на два отдельных пакета: только для чтения и только для записи. Программа в режиме чтения может запускать сколько угодно горутин и обращаться к любым полям структуры без ограничений, т.к. они просто читают память. В данном режиме не будет производиться никаких блокировок. В режиме записи будет работать только одна горутина на каждое поле, которая будет менять данные.

Если система будет находиться в состоянии чтения – писателей нет по определению. Если в записи – доступ только у одной горутины. Это приводит к тому, что условия для гонки данных никогда не будут выполняться. В данной архитектуре проблема соблюдения условий будет перекладываться на разработчика, если ошибиться и нарушить принципы архитектуры, то это гарантированно приведет к гонке данных. Так, например, если разработчик добавит изменение поля в пакете чтения, то вся архитектура будет нарушена, т.к. в пакете чтения нельзя изменять структуру хранилища. Именно поэтому нужен статический анализатор кода, который будет проверять соблюдение условий в написанном коде. Такой линтер смотрит на синтаксическое дерево, находит защищаемую структуру и пакет с чтением [2]. Если в данном пакете есть запись в какое-либо поле, то будет выдана ошибка и код просто не будет допущен до продакшена [3]. Одной из главных проблем является косвенное изменение через цепочки указателей. Для решения этой проблемы линтер строит граф всех зависимостей. Если находится вершина, которая ссылается на защищенную структуру, то все её дочерние вершины помечаются как потенциально опасные.

Выводы

Была доказана новая архитектура доступа к защищенной структуре и разработан статический анализатор кода, который проверяет соблюдения всех правил.

Литература

1. The Go Memory Model [Электронный ресурс]. – Режим доступа: https://go.dev/ref/mem (Дата обращения 28.11.2025).
2. Golang AST [Электронный ресурс]. – Режим доступа: https://pkg.go.dev/go/ast (Дата обращения 11.12.2025).
3. Golangci-lint [Электронный ресурс]. – Режим доступа: https://golangci-lint.run/ (Дата обращения 15.12.2025).