Programação Distribuida

Discussão sobre a entrega do projeto

Após implementar e testar sua solução, incluíndo cenários com múltiplos clientes compartilhando
múltiplas listas em simultâneo, escreva em alguns parágrafos quais as características e limitações
desse sistema em termos de escalabilidade, disponibilidade e consistência. Quais os pontos
possíveis de falha? Como o sistema lida com falhas (caso o faça)? Como a escalabilidade
poderia ser melhorada? A solução proposta para melhorar a escalabilidade causaria alguma
influência na consistência e/ou disponibilidade?

Escalabilidade:

- Imaginemos a situação hipotética de 10000 clientes tentarem acessar o server ou se uma lista possuir 500 milhões de itens, o sistema irá aguentar? Com base nisso, o servidor roda em um único processo, rodando em apenas uma unica máquina, toda a carga (as 4 operações) vão para esse processo, o map está com todas as listas alocadas na memoria ram dessa máquina, e todo acesso ao map possui um unico Mutex encarregado de controlar. Logo, analisando esse contexto de sobrecarga na RAM e de que a fila de execução vai gerar um gargalo, logo o sistema não vai aguentar.

Disponibilidade:

- Essa aplicação roda em um servidor, caso esse servidor trave ou a maquina pare de funcionar o serviço vai deixar de ser executado. A vantagem é a parte de recuperação que utiliza os Logs + o Snapshot, para carregar o estado atual da execução, no entanto se esse log for muito grande (Em caso de falha do snapshot), pode atrasar a inicialização do servidor, deixando ela lenta. Ficando mais tempo fora do ar.

Consistência:

- A aplicação possui consistência forte, por causa do Mutex, como só uma operação pode ser executada por vez, é impossivel que ocorra a leitura de um dado que está sendo inserido ou removido no meio de uma operação.

Por exemplo, quando o append libera o lock, o dado está salvo no map e no log, qualquer soliticação GET que vier depois vai ver o dado novo, daí surge o trade-off, para poder controlar esse acesso foi criado o gargalo com mutex, para ser mais consistente.

Pontos de falha:

- Se o servidor parar, o serviço para (Ponto unico de falha);
- O disco: Caso por alguma razão o log ou o snapshot corromper ou falhar, não é possivel recuperar o último estado do serviço.

Melhorias:

- Multiplos servidores, para resolver o problema do ponto unico (Replicação);
- Balanceamento de carga, para decidir para qual servidor mandar a requisição do cliente;
- Particionamento, dividir o map entre servidores, por exemplo, encarregar o servidor primário cuidar das listas de A-M e o secundário de N-Z. Para desafogar a RAM.

Influência:

- Com a ampliação dos servidores, ganha na escalabilidade e na disponibilidade;
- No entanto, um problema surge. A consistência forte some, por exemplo, Quando o cliente escrever no Servidor 1, como Servidor 2 vai saber que houve essa mudança de imediato? Sendo isso um problema clássico da sincronização de réplicas.