Universidade Presbiteriana Mackenzie

Faculdade de Computação e Informática - FCI

Projeto Prático 2 (LAB2)
Analisador de Desempenho de Cache de CDN com OpenMP

Computação Paralela

Prof. Dr. Jean M. Laine

1. Introdução (Hot Content)

1.1 O Cenário: A Aplicação Real (CDN)

Em sistemas modernos, a velocidade é tudo. Quando você acessa um serviço de streaming (como Netflix ou Globoplay) ou um grande portal de notícias, você não está se conectando a um único servidor. Você está se conectando a uma **CDN (Content Delivery Network)**, uma rede de servidores de cache distribuídos globalmente.
O objetivo da CDN é simples: manter cópias dos arquivos (vídeos, imagens, etc.) o mais perto possível de você, em um "servidor de borda"(edge server). Isso reduz a latência e a carga no servidor de origem (backend).
Mas o cache (em RAM ou SSD) é caro e limitado. A CDN precisa tomar decisões em tempo real:

- **Replicação de Cache (Hot Content):** Se um arquivo (ex: o novo episódio de uma série) está sendo acessado por milhões de pessoas simultaneamente, ele é um **"conteúdo quente"(hot content)**. O sistema precisa identificar isso em segundos e replicar este arquivo para mais servidores de borda para distribuir a carga.
- **Evicção de Cache (Cold Content):** Se um arquivo (ex: um filme antigo) não é acessado há dias, ele é um "conteúdo frio"(cold content). O sistema deve removê-lo do cache para abrir espaço para novos conteúdos.

O nosso projeto simula o **motor de análise** que alimenta essa tomada de decisão.

1.2 O Desafio: A Contenção

No Projeto 1 (LAB1), vimos como um mutex global (`pthread_mutex_t`) pode se tornar um gargalo de serialização. Neste projeto, elevamos o desafio. Vamos processar milhões de acessos de log em paralelo para atualizar uma Tabela Hash compartilhada.

O desafio real é: o que acontece no cenário de "hot content"? Quando 90% do tráfego (milhares de threads) tenta atualizar *o mesmo nó* da tabela (o contador do "episódio novo") simultaneamente? Se usarmos um lock global (`#pragma omp critical`), a análise será tão lenta que o servidor de origem já terá caído antes de o relatório ficar pronto.
Este projeto força você a comparar uma sincronização de **granularidade grossa** (lenta, inútil para a aplicação) com uma de **granularidade fina** (`#pragma omp atomic`), analisando o impacto real da **contenção de lock** no desempenho e, por consequência, na viabilidade da aplicação.

2. Objetivos do Projeto

Ao final deste projeto, você será capaz de:

- Implementar uma solução paralela utilizando as diretivas do **OpenMP** (como `#pragma omp parallel for`).
- Identificar condições de corrida no acesso a estruturas de dados complexas (Tabela Hash).
- Implementar e comparar diferentes estratégias de sincronização quanto à granularidade:
  - **Sincronização de Granularidade Grosseira** (`#pragma omp critical`).
  - **Sincronização de Granularidade Fina** (`#pragma omp atomic`).
- Entender, na prática, o que é **contenção de lock** e como ela impacta a escalabilidade.
- Analisar o desempenho de algoritmos paralelos sob diferentes distribuições de dados (uniforme vs. "hotspot").
- Conectar a análise de desempenho (tempo) com o objetivo da aplicação (tomada de decisão).

3. Descrição do Problema

Fase 1: Construção da Estrutura de Dados (Sequencial)

O programa deve primeiro ler um arquivo `manifest.txt` (contendo as URLs) e construir uma **Tabela Hash** em memória, usando a implementação fornecida (`hash_table.c` e `hash_table.h`).

```
// Estrutura do nó (definida em hash_table.h)
typedef struct CacheNode {
  char* url;
  long hit_count;
  struct CacheNode* next; // Para tratar colisões (encadeamento)
} CacheNode;

```

O `hit_count` de todos os nós começa em 0.

Fase 2: Processamento do Lote de Acesso (Sequencial e Paralelo)

Após a tabela estar construída, o programa deve ler um arquivo `access_log.txt` (milhões de linhas). Para cada linha do log, ele deve extrair a URL, buscar este nó na Tabela Hash (`ht_get`) e, se encontrado, incrementar o campo `hit_count` daquela URL.

Fase 3: Verificação de Corretude (Obrigatório)

Ao final do processamento, **cada** executável deve gerar um arquivo de saída chamado `results.csv`. Para isso, você **deve** usar a função fornecida:

```
// Adicionada em hash_table.h
void ht_save_results(HashTable* ht, const char* filename);

// Exemplo de uso no final do seu main:
ht_save_results(tabela_hash, "results.csv");
ht_destroy(tabela_hash);

```

Implementações Requeridas

Você deve implementar esta Fase 2 em três versões:

1. **`analyzer_seq.c`:** Versão sequencial pura.
2. **`analyzer_par_critical.c`:** Versão paralela com OpenMP que usa `#pragma omp critical` (granularidade grosseira).
3. **`analyzer_par_atomic.c`:** Versão paralela com OpenMP que usa `#pragma omp atomic update` (granularidade fina).

Verificação de Corretude: Todas as três versões devem produzir um arquivo `results.csv` **idêntico** ao arquivo de gabarito fornecido (após ordenação).

4. Etapas do Projeto

Etapa 1: Preparação do Ambiente e Dados

1. **Tabela Hash:** Estude a interface `hash_table.h` (incluindo a função `ht_save_results`).
2. **Arquivos de Dados:** Baixe o script `generate_cdn_data.py` (v2) no Moodle.
3. **Gere os dados:** Execute o script generate_cdn_data.py. Ele irá gerar (e/ou compactar em um .zip):
   - `manifest.txt`: A lista de URLs únicas.
   - `log_distribuido.txt`: Log com acessos uniformes (Baixa Contenção).
   - `log_concorrente.txt`: Log com acessos "hotspot"(Alta Contenção).
   - `gabarito_distribuido.csv`: A contagem correta de hits para o log distribuído.
   - `gabarito_concorrente.csv`: A contagem correta de hits para o log concorrente.
4. **Ambiente de Hardware (Requisito):** O projeto **deve** ser executado em uma máquina com **múltiplos núcleos** de CPU (pelo menos 4 núcleos são recomendados).
   - **Atenção:** Ambientes virtualizados gratuitos (como GitHub Codespaces, Replit, ou VMs de 1 vCPU) **não** permitirão que você observe o speedup do paralelismo. A sua análise de escalabilidade (Etapa 3) será prejudicada ou inválida. Use um computador local (notebook, desktop) ou um ambiente de laboratório adequado.
5. **Ambiente de Software e Makefile:** Compile com `gcc` e a flag `-fopenmp`. Crie um `Makefile` que gere os três executáveis (ex: `seq`, `par_critical`, `par_atomic`). O `make` deve compilar seu analisador (ex: `analyzer_seq.c`) junto com o `hash_table.c` fornecido.

Etapa 2: Implementação das Três Versões

Implemente as versões sequencial, paralela (critical) e paralela (atomic). Lembre-se de chamar `ht_save_results(ht, "results.csv")` no final de cada uma.

Etapa 3: Análise de Escalabilidade (Baixa Contenção)

Use **apenas** o arquivo `log_distribuido.txt`. Meça o tempo de execução (Speedup, Eficiência) das versões paralelas com $N = 1, 2, 4, 8$ threads e compare com o sequencial.

Etapa 4: Análise de Contenção (Alta Contenção)

Use **apenas** o arquivo `log_concorrente.txt`. Meça o tempo de execução das três versões (com $N = 8$ threads) e compare os tempos absolutos.

5. Formato da Entrega

A entrega será um arquivo `.zip` contendo:

1. **Código-Fonte:** Todos os arquivos `.c` (os três analisadores).
2. **Makefile**.
3. **Relatório (relatorio.pdf)**:
   - **Introdução:** Descrição do problema de CDN e das estratégias de sincronização (critical vs. atomic).
   - **Ambiente de Teste:** Detalhes da máquina (CPU, **#núcleos físicos**, Frequência, RAM, SO).
   - **Verificação de Corretude:** Um *print* mostrando a saída do `diff` entre seu `results.csv` (ordenado) e o `gabarito_distribuido.csv` (ordenado). (Não deve haver diferença).
   - **Resultados (Análise 1 - Escalabilidade):** A Tabela e o Gráfico de Speedup (da Etapa 3).
   - **Resultados (Análise 2 - Contenção):** A Tabela e o Gráfico de Barras (da Etapa 4).
   - **Resultados (Análise 3 - Aplicação):**
     - Usando o `gabarito_concorrente.csv`, gere um gráfico (ex: barras) mostrando as **"Top 10"URLs** mais acessadas.
     - Explique qual decisão de negócios (ex: replicação de cache) você tomaria como engenheiro de CDN ao ver esse gráfico.
   - **Conclusão e Análise:**
     - Analise o Gráfico 1 (Escalabilidade): Por que o `par_critical` não escalou bem? Por que o `par_atomic` escalou melhor? (Se seu speedup foi baixo ou inexistente, justifique com base no seu ambiente de hardware).
     - Analise o Gráfico 2 (Contenção): O que é **contenção de lock**? Como o "hotspot"afetou cada versão?
     - Analise o Gráfico 3 (Aplicação): Por que a versão `par_critical` (lenta) tornaria a sua decisão de negócios (replicação) inútil no mundo real?

6. Avaliação

A nota final será composta com base nos critérios abaixo:

Critério	Descrição	Peso
Corretude e Funcionalidade	O código compila com o Makefile. As três versões produzem um `results.csv` **idêntico** ao gabarito fornecido (verificação de corretude).	3,0 pts
Implementação Paralela	Uso correto das diretivas OpenMP. Implementação correta das estratégias `critical` (grosseira) e `atomic` (fina) nos locais adequados.	4,0 pts
Relatório e Análise	Qualidade e completude da documentação. O relatório contém tudo que foi solicitado? Possui clareza e coerência nas tabelas e nos três gráficos (Speedup, Contenção, Top 10)? Profundidade da análise na conclusão, explicando o *porquê* do desempenho (granularidade, contenção) e conectando-o com o *impacto* na aplicação real.	3,0 pts

7. Regulamento de Apresentação, Defesa, Originalidade e Entrega dos Projetos Práticos

1 Apresentação

1.1. A apresentação do projeto é obrigatória para a atribuição de nota.

1.2. Todos os integrantes do grupo devem estar presentes durante a apresentação, sendo responsáveis pela defesa do trabalho.

1.3. A ausência de qualquer integrante no momento da apresentação acarretará nota zero no projeto para o aluno ausente.

1.4. Caso o grupo inteiro não compareça na data agendada, o trabalho será considerado não apresentado e receberá nota zero.

2 Defesa do Código

2.1. Durante a apresentação, cada aluno deve demonstrar domínio sobre o código-fonte, sendo capaz de:

- Explicar a lógica de funcionamento de qualquer parte da implementação;
- Justificar as decisões de estrutura e algoritmo utilizadas;
- Realizar alterações ou correções solicitadas pelo professor no momento da apresentação, se necessário.