Universidade do Minho

Braga, Portugal

TRABALHO PRÁTICO - RELATÓRIO

Armazenamento de dados em memória com acesso remoto

Sistemas Distribuídos

Departamento de Informática

Engenharia Informática 2024/25

Equipa de Trabalho:

A104365 - Fábio Magalhães

A104080 - João Macedo

A104185 - Filipe Fernandes

A104267 - André Pinto

Índice

1. Introdução

Neste relatório, apresenta se o resultado do trabalho prático realizado no contexto da Unidade Curricular de Sistemas Distribuídos. O sistema, desenvolvido em Java, implementa um serviço de armazenamento de dados partilhado, no qual a informação é armazenada em memória num servidor e acedida remotamente por múltiplos clientes de forma concorrente. Os clientes comunicam com o servidor através de sockets TCP, de forma a inserir e/ou consultar informações.

Ao longo do relatório, descreve-se a arquitetura do sistema, justificando alguns dos aspetos mais relevantes na concretização do sistema desenvolvido. Além disso, apresenta-se os resultados dos cenários de teste concebidos de forma a avaliar a eficiÊncia e a escalabilidade do sistema em diferentes cenários de utilização.

2. Arquitetura do Sistema e Protocolos

2.1. Servidor

O servidor implementa um sistema de comunicação cliente-servidor, onde para cada cliente é inicializado um `ServerWorker` que processa comandos recebidos através de uma `FramedConnection`. O servidor suporta operações de autenticação (`LOGIN`) e registo de clientes (`REGISTER`), bem como operações simples e compostas de inserção, remoção e consulta de dados. O servidor conta ainda com dois gestores, um `AccountManager` para a autenticação de utilizadores, e um `DataManager`, para a implementação de operações de armazenamento de dados num formato chave-valor.

2.2. Cliente

O cliente estabelece uma ligação com o servidor através de um socket TCP e comunica com ele de forma assíncrona. Quando se estabelece a ligação, o cliente envia o comando `CONNECT` ao servidor . Em paralelo, o programa utiliza duas threads principais: uma responsável por receber as respostas do servidor e outra que processa as entradas do utilizador. A thread responsável pelas respostas do servidor fica à escuta cotinuamente, aguardando mensagens do servidor, que são analisadas e exibidas conforme o tipo. A thread de entradas do utilizador fica à espera de comandos e envia-os para o servidor em novas threads, assegurando que a execução do programa não seja bloqueada. A utilização dessas duas threads permite que o cliente envie e receba dados simultaneamente, sem que uma operação interfira na outra.

2.3. AccountManager

O servidor possui um gestor de contas responsável por gerir as contas de utilizadores e os clientes nele ligados. Ele mantém uma lista de contas (`accounts`) e outra de clientes com uma ligação estabelecida (`connected_clients`). O gestor de contas também gere a adição e remoção de clientes, garantindo que o número de ligações não ultrapasse o limite máximo definido.

2.4. DataManager

O servidor possui ainda outro gestor responsável por gerir os dados nele armazenados em memória. Ele mantém um map (`data`) que associa chaves a valores, e implementa métodos de inserção, remoção e consulta de dados. Além disso, implementa operações de escrita e leitura compostas, como adicionar/remover múltiplos pares de dados de uma só vez, e uma operação de leitura de dados condicional, que fica a aguardar até que as condições sejam satisfeitas.

Recorrendo ao `ReentrantLock`, garante-se que as operações de escrita e leitura de dados sejam operações atómicas, de forma a garantir a integridade dos dados e a evitar que ocorram condições de corrida.

3. Análise de Resultados

```
BenchmarkScenario(description:"Read-heavy workload", readPercentage:70, iterations:500),
BenchmarkScenario(description:"Write-heavy workload", readPercentage:30, iterations:500),
BenchmarkScenario(description:"Balanced workload", readPercentage:50, iterations:500)
```

Figura 1: Ficheiro de configuração dos 3 `benchmarks` de teste

Neste primeiro teste foram realizados 3 cálculos temporais de cerca de `500` iterações em diferentes `workloads`:

- `Read-heavy`: Cerca de 70% das operações eram de leitura e as restantes de escrita (escrita pesada)
- `Write-heavy`: Cerca de 30% das operações eram de leitura e as restantes de escrita (leitura pesada)
- `Balanced`: Cerca de 50% das operações eram de leitura e as restantes de escrita (leitura e escrita balanceadas)

e os resultados foram os seguintes:

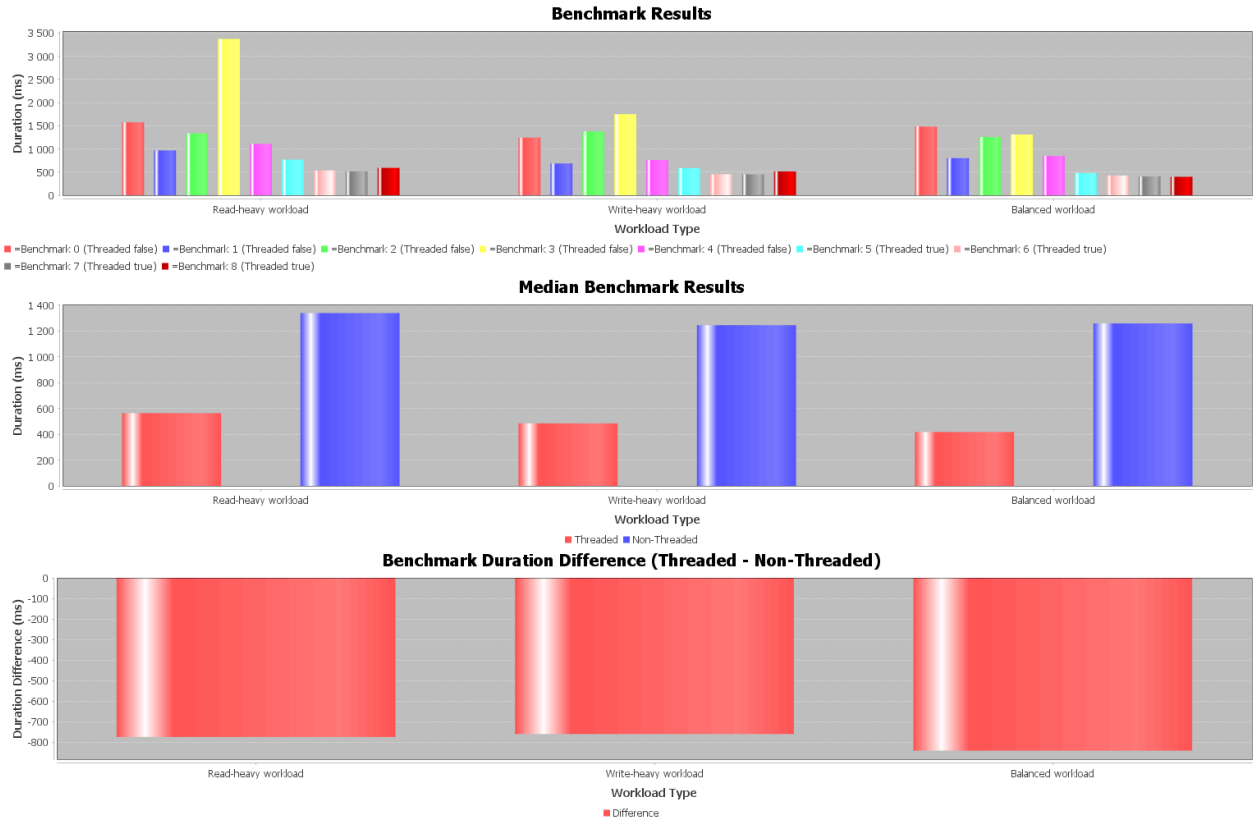

Figura 2: Gráficos com os resultados do teste realizado de 500 iterações

Os gráficos acima apresentados ilustram o desempenho do sistema de acordo com diferentes cenários de utilização (workloads). Pode-se observar que, em média, as operações concorrentes realizadas por threaded clients apresentam tempos de resposta significativamente menores em comparação com operações sequenciais. Este comportamento evidencia a capacidade da

abordagem multi-threaded de explorar o paralelismo para lidar, de forma eficiente, com múltiplas requisições concorrentes. Além disso, nota-se que um workload balanceado em ambientes multi-threaded resulta em tempos de execução bastante mais rápido, o que indica uma boa distribuição de tarefas entre as várias threads e a capacidade do sistema lidar eficientemente com cargas equilibradas.

No próximo teste realizado foi aumentado o número de iterações para `2000` para verificar consistência nos resultados obtidos anteriormente e o desempenho do programa conforme o aumento da escalabilidade.

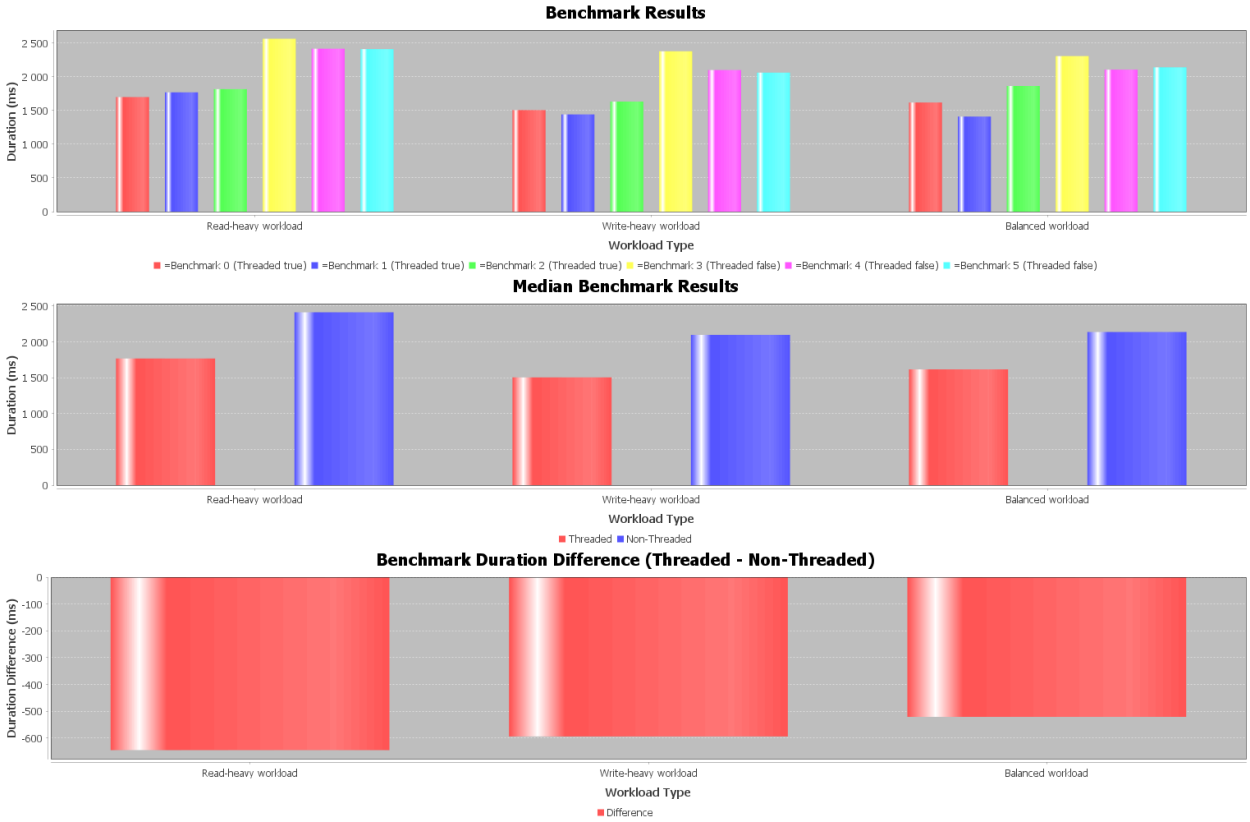

Figura 3: Gráficos com os resultados do teste realizado de 2000 iterações

Os gráficos demonstram que o comportamento observado em operações concorrentes e realizadas por threaded clients se mantém mais rápido em relação às operações non-threaded. No entanto, também é evidente que as operações non-threaded se tornam ainda mais lentas à medida que a carga aumenta. Curiosamente, em workloads balanceados, a diferença entre operações threaded e non-threaded é consideravelmente menor do que no cenário anterior. Este comportamento sugere que, com o aumento do número de iterações, os desempenhos das operações threaded e non-threaded tenderiam a se aproximar.

4. Conclusão

O trabalho realizado permitiu-nos compreender e aplicar na prática diversos conceitos fundamentais no desenvolvimento de sistemas distribuídos. Através da implementação de um sistema distribuído de armazenamento de dados em memória, foi possível explorar aspetos cruciais como a comunicação cliente-servidor através de sockets TCP, a gestão de múltiplos clientes de forma concorrente e a utilização de primitivas de sincronização para aceder a recursos partilhados.

Assim, este projeto permitiu-nos consolidar os conhecimentos adquiridos ao longo da unidade curricular de Sistemas Distribuídos, bem como desenvolver competências de programação em Java e de resolução de problemas complexos, que são cruciais para o desenvolvimento de sistemas de software robustos e eficientes.