SUPSI

cpp-consistent-hashing-algorithms

Semester Project

University of Applied Sciences and Arts of Southern Switzerland (SUPSI)
Department of Innovative Technologies (DTI)
Institute of Information Systems and Networking (ISIN)

Student: Roberto Vicario

August 9, 2024

Overview

Within the scope of a research project, conducted under the guidance of Massimo Coluzzi, a framework in Java was developed for benchmarking state-of-the-art consistent hashing algorithms.

In order to explore the performance of these algorithms using different languages, I was tasked with creating a version in C++. This version currently encompasses only a partial selection of the technologies considered in the ISIN framework.

Overview

However, the tool has been designed adaptively to easily implement new algorithms and benchmarks. The software remains consistent with the behavior of its Java counterpart. It is designed to operate via a command-line interface and to save benchmark results in a CSV file, utilizing YAML files for parameter configuration.

Algoritmhs

All the algorithms were implemented in the `Algorithms/` package, with each algorithm having its own dedicated subpackage within it.

Implemented:

- **Jump** by Lamping et al. (2014)
- **Anchor** by Mendelson et al. (2020)
- **Dx** by Dong et al. (2021)
- **Power** by Leu et al. (2023)
- **Memento** by Coluzzi et al. (2023)

Algoritmhs

Example in `Main.cpp`

```
/*
 * ANCHOR
 */
auto capacity = i["args"]["capacity"].as<int>(10);
execute<AnchorEngine>(*handler, yaml, "anchor", capacity);
```

Algoritmhs

Some engines are not yet implemented in this project but are already configured for execution in the benchmark routine:

- **Ring** by Karger et al. (1997)
- **Rendezvous** by Thaler et al. (1998)
- **Multi-probe** by Appleton et al. (2015)
- **Maglev** by Eisenbud et al. (2016)

Algoritmhs

Example in `Main.cpp`

```
/*
 * MAGLEV
 */
// auto permutations = i["args"]["permutations"].as<int
    >(128);
// execute<MaglevEngine>(handler, yaml, "maglev",
    permutations);
```

Benchmarks

As outlined in *"Consistently Faster: A Survey and Fair Comparison of Consistent Hashing Algorithms." by Coluzzi et al. (2023)* [1], here is a concise overview of the benchmarks utilized:

- **Balance:** Distribution of keys across nodes.
- **Initialization Time:** Time to set up internal structures.
- **Lookup Time:** Time to locate the node for a given key.
- **Memory Usage:** Memory used by the algorithm.
- **Monotonicity:** Resource movement on cluster scaling.

Benchmarks

- **Resize Balance:** Balance after adding/removing nodes.
- **Resize Time:** Time to reorganize after cluster changes.

Configuration

The format of the configuration file is described in detail in the `configs/template.yaml` file. The tool will use the `configs/default.yaml` file that represents the default configuration if no configuration file is provided.

(!) Important

Each algorithm and benchmark adjusts its behavior based on parameters set in a YAML configuration file. This file also outlines how to configure the benchmark routine, including details like the number of iterations, initialization nodes to use, and the hash function for each algorithm.

Control Flow

The program begins execution in `Main.cpp`, which serves as the entry point. This file invokes `Handler/HandlerImpl.hpp`, which is responsible for creating a CSV file to store benchmark results, updating the file regularly.

Next, `Main.cpp` reads the YAML configuration and initiates the routine for each algorithm.

Control Flow

For each algorithm, `Main.cpp` calls a method named `execute`, which is implemented in `Benchmarks/Routine.hpp`. This file contains the routine that iterates over benchmarks, hash functions, and initialization nodes for each iteration.

Finally, the handler updates the data in the CSV file, flushes the stream, continues processing all benchmarks, and closes the stream when finished.

Control Flow

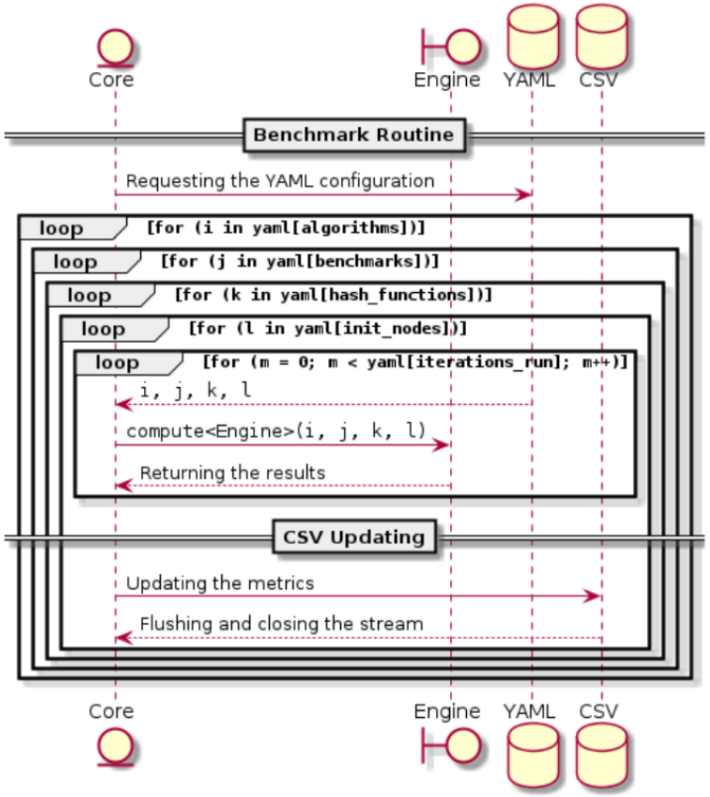

Figure: A UML sequence diagram to explain how the benchmark routine procedure works.

Instructions

1. Clone the repository and navigate to the cloned repository:

```
$ git clone https://github.com/robertovicario/
cpp-consistent-hashing-algorithms.git
$ cd cpp-consistent-hashing-algorithms
```

2. Run repository setup:
   - vcpkg:

```
# Ensure scripts have executable permissions:
# chmod +x repo.sh
$ ./repo.sh
```

   - CMake:

```
# Ensure scripts have executable permissions:
# chmod +x cmake.sh
$ ./cmake.sh
```

Instructions

3. Build the project with Ninja:

```
$ cd build
$ ninja
```

4. Start the framework:
   - Default configuration:

     ```
     $ ./main
     ```

   - Custom configuration:

     ```
     $ ./main <your_config>.yaml
     ```

5. Navigate to `build/tmp/` and check the `results.csv` file.

Adding new Algorithms

1. Insert the algorithm name into any configuration file located in `configs/`.
2. Implement your algorithm in `Algorithms/your_algo/`. Keep in mind that the system employs C++ templates to integrate the algorithms into the loop.

Adding new Algorithms

3. Integrate a new execution routine into `Main.cpp`. Append a new `else if` branch and incorporate your engine using:

Example in `Main.cpp`

```
/*
 * NEW_ALGORITHM
 */
execute<YourEngine>(handler, yaml, "your_algo");
```

Adding new Algorithms

If your engine requires additional parameters, include them as follows:

Example in `Main.cpp`

```
/*
 * NEW_ALGORITHM
 */
execute<YourEngine>(handler, yaml, "your_algo",
param1, param2, ..., paramN);
```

Adding New Benchmarks

1. Insert the benchmark name into any configuration file located in `configs/`.
2. Implement the benchmark in `Benchmarks/`. Note that the system utilizes C++ templates for benchmark integration into the loop.

Adding New Benchmarks

3. Integrate a new benchmark routine into `Benchmarks/Routine.hpp`. Append a new `else if` branch and incorporate your benchmark using:

Example in `Routine.hpp`

```
/*
 * NEW_BENCHMARK
 */
printInfo(l, algorithm, benchmark, hashFunction,
initNodes, iterationsRun);
results[l] = computeYourBenchmark<Engine>(yaml, algorithm,
initNodes, args...);
```

License

Credits

- **java-consistent-hashing-algorithms:**
  - **Author:** SUPSI-DTI-ISIN
  - **License:** GNU General Public License version 3
  - **Source:** GitHub Repository
- **cpp-anchorhash:**
  - **Author:** anchorhash
  - **License:** The MIT License
  - **Source:** GitHub Repository
- **DxHash:**
  - **Author:** ChaosD
  - **License:** none
  - **Source:** GitHub Repository

Contacts

- **Supervisor:** Amos Brocco @slashdotted
- **Student:** Roberto Vicario @robertovicario

Bibliography

Massimo Coluzzi, Amos Brocco, and Tiziano Leidi.
Consistently faster: A survey and fair comparison of consistent hashing algorithms.
In *SEBD*, pages 51–64, 2023.