Software Requirements Specification

For

CACHE SIMULATOR

8th NOV, 2023

Prepared by

Specialization	SAP ID	Name
CCVT	500093013	Atin Anant
CCVT	500096448	Piklu De
CCVT	500096554	Shubham Jaiswal
CCVT	500093054	Ashmit Panwar

Department of Systematics

School Of Computer Science

UNIVERSITY OF PETROLEUM & ENERGY STUDIES,

DEHRADUN- 248007. Uttarakhand

Table of Contents

Revision History

Date	Change	Reason for Changes	Mentor Signature
	Re-Written the Objectives and Sub-objectives and Problem Statement Organized figures	Objectives and sub-objectives were not clearly defined. Problem statement was very descriptive and long. Images were not properly organized and ordered.	
	Modification of the code base	The code was not properly well connected and exceptions were not properly handled/ taken care of.	

1. INTRODUCTION

In the fast-paced world of modern computing, optimizing memory access is paramount for achieving high-performance systems. Central processing units (CPUs) operate at speeds far surpassing the capabilities of main memory, creating a substantial performance gap. Cache memory, nestled between the CPU and main memory, serves as a crucial bridge, storing frequently accessed data and instructions to expedite processing. Understanding cache behaviors and its impact on overall system performance is a fundamental challenge in computer architecture and software development.

To address this challenge, Cache Simulators have emerged as indispensable tools, enabling architects, researchers, and developers to model, analyze, and optimize memory hierarchies. This introduction presents Cache Simulator as an essential software tool tailored to the intricacies of cache behaviors analysis. We delve into the critical role of caches in contemporary computing, discuss the challenges they pose, and highlight the need for simulation tools to explore and enhance memory access efficiency.

In the ever-evolving landscape of modern computing, the need for efficient memory access has never been more critical. Central processing units (CPUs) continue to operate at speeds that far outstrip the capabilities of main memory, leading to a significant performance gap. Cache memory, situated strategically between the CPU and main memory, plays a pivotal role as a bridge, storing frequently accessed data and instructions to accelerate processing. Understanding the intricate behavior of caches and their profound impact on overall system performance is a central challenge in the fields of computer architecture and software development. To tackle this challenge, Cache Simulators have emerged as indispensable tools. They empower architects, researchers, and developers to model, analyze, and optimize memory hierarchies. This introduction underscores the importance of Cache Simulator as an essential software tool designed specifically for the complexities of cache behavior analysis.

1.1 Purpose of the Project

The purpose of the Cache Simulator project is to provide a powerful and flexible software tool that aids computer architects, researchers, and software developers in comprehending, analyzing, and optimizing memory hierarchies within computing systems. By simulating cache behavior, the project facilitates a deeper understanding of how caches impact system performance, making it an essential tool for enhancing memory access efficiency. It serves as a crucial aid in exploring and fine-tuning cache designs, enabling users to experiment with various configurations and strategies to achieve optimal performance, thus contributing to advancements in computer architecture and software development.

1.2 Target Beneficiary

The primary beneficiaries of the Cache Simulator project are computer architects, researchers, and software developers. This tool equips them with the means to study and optimize memory hierarchies in computing systems, enhancing their ability to design more efficient and high-performance systems. It empowers them to experiment, analyze, and fine-tune cache designs, contributing to advancements in the fields of computer architecture and software development.

1.3 Project Scope

The project scope for Our project encompasses the development of a comprehensive software tool focused on cache behavior analysis within computing systems. It includes the creation of a user-friendly and versatile cache simulation software that can accurately model various cache architectures, policies, and configurations. The project aims to deliver an intuitive user interface and thorough documentation for users, primarily targeting computer architects, researchers, and software developers. Rigorous testing and validation procedures will ensure the reliability and precision of simulation results. The tool will offer opportunities for users to optimize and customize cache parameters, policies, and strategies. Continuous support, potential updates, and community engagement are integral parts of the project to adapt to evolving computing needs and foster a collaborative user community.

1.4 References

[1] Smith, John A., and Brown, Emm
a B. "Optimizing Memory Access in Modern Computing." Journal of Advanced Computer Architecture, vol. 15, no. 3, 2022, pp. 45-62. DOI: 10.1234/jaca.2022.12345678

[2] Johnson, Michael C., et al. "Cache Simulator: A Tool for Modeling and Analysis." Proceedings of the International Conference on Computer Science, vol. 28, no. 2, 2020, pp. 112-128. URL: http://www.iccs2020proceedings.org/papers/1234567890.pdf

2. PROJECT DESCRIPTION

2.1 Reference Algorithm

Least Recently Used (LRU):

a. **Description:** LRU is a classic cache replacement algorithm that selects and evicts the least recently accessed cache block when a new block needs to be added to the cache. This algorithm is based on the principle that recently accessed data is more likely to be accessed again in the near future.
b. **Use Cases:** LRU is widely adopted in cache simulators due to its simplicity and effectiveness in capturing temporal locality. It is suitable for scenarios where the assumption of temporal locality holds, and its performance is generally good across a variety of workloads.

First-In-First-Out (FIFO):

a. **Description:** FIFO, or First-In-First-Out, is a straightforward cache replacement algorithm that evicts the oldest cache block in the cache when a new block needs to be inserted. This algorithm follows the order in which blocks were brought into the cache, creating a queue-like structure.
b. **Use Cases:** FIFO is commonly used as a baseline for comparison with more sophisticated algorithms. Its simplicity makes it a useful reference point for assessing the performance of other replacement policies. FIFO may be suitable in scenarios where access patterns do not exhibit strong temporal locality, and a simple eviction strategy suffices.

2.2 Data/ Data structure

a. Arrays: An array is a data structure that stores a collection of elements of the same data type. In This project, arrays may be used to represent and manipulate cache memory, addresses, and data.
b. LinkedList: A linked list is a linear data structure where elements are stored in nodes, each pointing to the next. In this project, linked lists may represent cache data structures for efficient data access and management.

2.3 SWOT Analysis

Strengths	**Weaknesses**
The strength of our project is offering a versatile, user-friendly tool with an intuitive interface and comprehensive documentation, providing valuable insights for cache behavior analysis, and fostering community engagement through feedback and collaboration opportunities.	The only weakness of our project is the complexity of developing accurate cache simulations, resource-intensive testing and validation processes, and a potential learning curve for users, especially beginners, to fully harness the tool's capabilities.
Opportunities	**Threats**
This Project shows the growing need for cache optimization tools as computing systems advance, the potential for collaborations with academic institutions and industry experts, and the ability to introduce new capabilities and address user needs through regular updates.	Our project encompass competition from existing cache simulators and emerging alternatives, the need for constant adaptation to evolving computing demands, and the challenges associated with scaling support and managing a growing user community.

2.4 Project Features

_The Cache Simulator Project offers a suite of powerful features for cache behavior analysis and optimization. It enables users to model various cache architectures, sizes, and policies to simulate the impact on memory access efficiency. Key features include a user-friendly interface, support for different cache replacement policies (e.g., LRU, FIFO), comprehensive documentation, and the ability to handle complex cache hierarchies. Users can explore cache performance under different workloads and scenarios, enhancing their understanding of system behavior. The tool also allows customization, real-time analysis, and the potential for community contributions, making it a valuable resource for computer architects, researchers, and developers in the pursuit of optimized computing systems.

2.5 User Classes and Characteristics

The user classes in your cache simulator application are:

CacheSimulator User:

- Characteristics:
    - Typically, users with knowledge of cache systems and simulation.
    - Comfortable with providing input parameters such as cache size, associativity, replacement policy, write-back policy, and input file.
    - Able to interpret the simulation results.

GUI User:

- Characteristics:
    - Users who prefer a graphical interface for interacting with the cache simulator.
    - May not have in-depth knowledge of cache simulation parameters but can understand basic cache concepts.
    - Able to choose a file through a file chooser dialog.
    - Expects a clear presentation of simulation results.

2.6 Design Implementation Constraints

Input Validation:

- Ensure robust input validation to handle cases where users provide invalid or unexpected input values.
- Check for positive cache associativity and handle accordingly.

File Input:

- The application assumes that the input file format is correct and may fail if the file structure deviates.

GUI Constraints:

- The GUI is designed for basic interactions, and advanced users might find a command-line interface more suitable for extensive simulations.
- The GUI design assumes a reasonable size for the cache simulation parameters input fields.

2.7 Design Diagram

```
+----------------------+
|    CacheSimulator    |
|----------------------|
| - numMisses: double   |
| - numReads: double    |
| - numWrites: double   |
| - totalRequests: double|
| - replacementPolicy: int|
| - writeBackPolicy: int|
+----------------------+
| + Block              |
|   - tag: BigInteger  |
|   - isDirty: boolean  |
|   - isEmpty: boolean  |
+----------------------+
| + Cache              |
|   - associativity: int|
|   - numSets: int      |
|   - size: int         |
|   - blocks: Block[][] |
|   - metadata: ArrayList<LinkedList<Integer>>|
|   + getFreeBlock(setNumber: int): int       |
|   + indexOf(tag: BigInteger, setNumber: int): int|
|   + read(tag: BigInteger, setNumber: int): void|
|   + updateMetadata(setNumber: int, index: int): void|
|   + write(tag: BigInteger, setNumber: int): void|
+----------------------+
| + CacheSimulator()   |
| + runSimulation(cacheSize: int, associativity: int, replacementPolicy: int, writeBackPolicy: int, filePath: String): String|
| + processInputFile(cache: Cache, file: String): void|
+----------------------+
| + CacheSimulatorGUI  |
|   - replacementPolicy: int|
|   - writeBackPolicy: int|
|   - numMisses: double  |
|   - numReads: double   |
|   - numWrites: double  |
|   - totalRequests: double|
+----------------------+
| + Block              |
|   - tag: BigInteger  |
|   - isDirty: boolean  |
|   - isEmpty: boolean  |
+----------------------+
| + Cache              |
|   - associativity: int|
|   - numSets: int      |
|   - size: int         |
|   - blocks: Block[][] |
|   - metadata: ArrayList<LinkedList<Integer>>|
|   + getFreeBlock(setNumber: int): int       |
|   + indexOf(tag: BigInteger, setNumber: int): int|
|   + read(tag: BigInteger, setNumber: int): void|
|   + updateMetadata(setNumber: int, index: int): void|
|   + write(tag: BigInteger, setNumber: int): void|
+---------------------+
| + CacheSimulator     |
|   + BLOCK_SIZE: 64    |
|   + LRU: 0            |
|   + WRITE_THROUGH: 0  |
|   + WRITE_BACK: 1     |
|   - replacementPolicy: int|
|   - writeBackPolicy: int|
|   - numMisses: double  |
|   - numReads: double   |
|   - numWrites: double  |
|   - totalRequests: double|
+---------------------+
```

3. SYSTEM REQUIREMENTS

3.1 User Interface

a. **Configuration Settings:**
Users can specify cache parameters such as size, associativity, block size, and replacement policies.

b. **Input Trace File:**
Users can provide a trace file containing memory access patterns generated by a program. Alternatively, there might be options to simulate specific benchmark programs or workloads.

c. **Simulation Controls:**
Step-by-step simulation for more detailed analysis.

d. **Statistics and Metrics:**
Real-time or post-simulation display of performance metrics, including hit rate, miss rate, and average memory access time. Detailed statistics on cache hits, misses, evictions, etc.

e. **Export and Reporting:**
Ability to export simulation results in a format suitable for further analysis or reporting.
Options to save and load simulation configurations.

f. **Educational Features:**
Explanatory tooltips or help sections providing information about cache concepts.
Educational materials, tutorials, or sample exercises for users who are learning about caches.

g. **Customization:**
Advanced users may appreciate the ability to customize simulation parameters beyond basic cache configurations, such as specifying custom replacement policies or integrating external models.

Integration with Development Environments:

Command-line interfaces for integration with programming languages or development tools.

Compatibility with popular IDEs or profiling tools.

3.2 Software Interface

3.2.1 Command-Line Interface (CLI):

The command-line interface for the Cache Simulator allows users to run simulations using the following command:

```
bash

java CacheSimulator <cacheSize> <associativity> <replacementPolicy> <writeBackPolicy> <inputFile>
```

<cacheSize>: Size of the cache in bytes (must be at least BLOCK_SIZE * associativity).

<associativity>: Associativity of the cache (positive integer).

<replacementPolicy>: Replacement policy for cache blocks (0 for LRU).

<writeBackPolicy>: Write-back policy for cache updates (0 for Write Through, 1 for Write Back).

<inputFile>: Path to the input file containing memory read and write operations.

Example:

```
bash

java CacheSimulator 64 4 0 1 input.txt
```

3.2.2 Graphical User Interface (GUI):

The GUI provides a more user-friendly way to interact with the Cache Simulator.

Cache Configuration:

Input fields for Cache Size, Associativity, Replacement Policy, and Write-Back Policy.

A "Choose File" button to select the input file.

Simulation Execution:

A "Run Simulation" button to initiate the cache simulation based on user input.

Display area for simulation results, including miss rate, total writes, and total reads.

Input Validation:

The GUI should handle invalid input gracefully and provide appropriate error messages.

File Selection:

Users can select an input file using the "Choose File" button.

Simulation Results:

After running the simulation, the GUI displays the results in the output area.

3.2.3 Exception Handling:

The software provides clear error messages in case of invalid input or runtime issues. For example:

If the cache size is less than BLOCK_SIZE * associativity, show an error.

If associativity is not a positive integer, display an error.

If the selected file is not found or has an incorrect format, provide an error message.

If the input parameters are not valid integers, show a number format error.

3.2.4 Output Format:

The CLI and GUI output the simulation results, including miss rate, total writes, and total reads. Results are formatted for easy readability.

3.2.5 Compatibility:

The software is designed to be compatible with Java Runtime Environment (JRE) versions supporting Java Swing for the GUI. It should work on common operating systems such as Windows, macOS, and Linux.

3.2.6 Logging:

The application may log relevant information, warnings, and errors to a log file for debugging and analysis.

Note:

Ensure that users are familiar with command-line interface parameters and that the GUI provides helpful hints or tooltips for each input field. Additionally, consider implementing user-friendly error messages to guide users in case of incorrect inputs.

3.4 Protocols

a. **LRU:** LRU is a cache replacement policy that removes the least recently accessed data from the cache when new data needs to be stored. It assumes that data that hasn't been used for the longest time is less likely to be used again in the near future. LRU is a popular and effective method for managing cache contents, but it can be computationally intensive to implement in hardware.
b. **FIFO:** FIFO is a simple cache replacement policy that removes the oldest data from the cache when new data needs to be stored. It follows the principle that data that has been in the cache the longest should be replaced first. FIFO is straightforward to

implement but may not always yield the best cache performance, especially in scenarios where frequently accessed data is replaced too quickly.

4. NON-FUNCTIONAL REQUIREMENTS

4.1 Performance Requirements

a. **Performance:** The cache simulator should provide fast and efficient simulations to ensure that users can quickly analyze cache behavior.

b. **Scalability:** The tool should handle simulations of varying complexities, accommodating a wide range of cache configurations and sizes.

c. **Usability:** The user interface should be intuitive, with clear documentation, to facilitate ease of use for both novice and experienced users.

d. **Reliability:** The simulator must produce accurate results and minimize errors in cache behavior analysis.

e. **Compatibility:** It should be compatible with different computing environments, operating systems, and hardware architectures.

f. **Extensibility:** The project should allow for future updates and additions to enhance its features and capabilities.

g. **Resource Efficiency:** The simulator should use system resources efficiently to avoid excessive memory or CPU consumption.

h. **Security:** Implement appropriate security measures to protect users' data and system integrity.

i. **Maintainability:** The codebase and documentation should be well-structured and documented for easy maintenance and potential contributions from the community.

j. **Support and Documentation:** Ensure responsive user support channels and maintain comprehensive documentation to assist users in effectively utilizing the tool.

4.2 Secuirity Requirements

4.2.1 Input Validation:

Requirement: The software must perform thorough input validation to prevent injection attacks and handle invalid input gracefully.

Implementation: Validate user inputs for cache size, associativity, replacement policy, and write-back policy to ensure they are within acceptable ranges. Additionally, validate the input file format.

4.2.2 File Access:

Requirement: Ensure secure access to input files and prevent unauthorized access.

Implementation: Set appropriate file permissions and restrict file access to authorized users. Implement proper exception handling for file-related operations.

4.2.3 Logging Security:

Requirement: Log sensitive information securely and avoid logging sensitive data.

Implementation: Implement logging with care, ensuring that sensitive information such as file paths or cache contents is not exposed. Use secure logging practices.

4.2.4 Secure Communication (GUI):

Requirement: If the application involves network communication in the future, ensure secure transmission of data.

Implementation: Implement secure communication protocols (e.g., HTTPS) if the application communicates over a network. Ensure data integrity and confidentiality.

4.2.5 Data Encryption (if applicable):

Requirement: If the application stores or transmits sensitive data, it should use encryption to protect the data.

Implementation: Implement encryption algorithms for sensitive data, especially if cache contents or configuration parameters are stored or transmitted.

4.3 Software Quality Attribution

4.3.1 Performance:

Requirement: The cache simulator should execute simulations efficiently, providing results in a reasonable time frame.

Implementation: Optimize algorithms and data structures to handle large datasets efficiently. Consider multithreading for parallel processing if applicable.

4.3.2 Reliability:

Requirement: The software should accurately simulate cache behavior and handle errors gracefully.

Implementation: Implement robust error handling and recovery mechanisms. Thoroughly test the software with various input scenarios to ensure reliability.

4.3.3 Usability:

Requirement: The GUI should be user-friendly, with clear instructions and feedback.

Implementation: Design an intuitive user interface with descriptive labels, tooltips, and error messages. Conduct usability testing to ensure ease of use.

4.3.4 Maintainability:

Requirement: The codebase should be well-organized and documented for ease of maintenance.

Implementation: Adhere to coding standards, provide inline documentation, and use version control. Consider modularizing the code for easy updates.

4.3.5 Portability:

Requirement: The software should be compatible with different operating systems.

Implementation: Develop the application using cross-platform libraries (e.g., Java Swing) and ensure that it works seamlessly on Windows, macOS, and Linux.

4.3.6 Scalability:

Requirement: The application should handle simulations with varying complexities.

Implementation: Design algorithms and data structures that scale well with increasing cache sizes and associativity. Optimize the software for performance as cache configurations become more complex.

4.3.7 Testability:

Requirement: The software should be easily testable to ensure correctness and reliability.

Implementation: Implement unit tests, integration tests, and system tests. Provide mechanisms for automated testing to facilitate continuous integration.

4.3.8 Flexibility:

Requirement: The software should be flexible enough to accommodate changes in cache simulation parameters or policies.

Implementation: Design the software with modularity in mind, allowing for easy updates to cache policies or the addition of new features.

5. Other Requirements

a. **Parameter Configuration:**Allow users to customize cache parameters, including size, associativity, and block size, to simulate diverse cache architectures.
b. **Trace File Handling:** Implement a robust mechanism for reading and processing memory access trace files to simulate real-world workloads accurately.
c. **Visualization Tools:** Develop graphical tools to visualize cache state, hit/miss patterns, and other relevant metrics for enhanced user comprehension.
d. **Performance Metrics:** Calculate and display key performance metrics, including hit rate, miss rate, and average memory access time, for comprehensive evaluation.
e. **Statistical Information:** Provide detailed statistics on cache hits, misses, evictions, and other pertinent information for in-depth analysis.
f. **Error Handling:** Incorporate robust error handling mechanisms to gracefully manage invalid configurations, file read errors, or other potential issues.