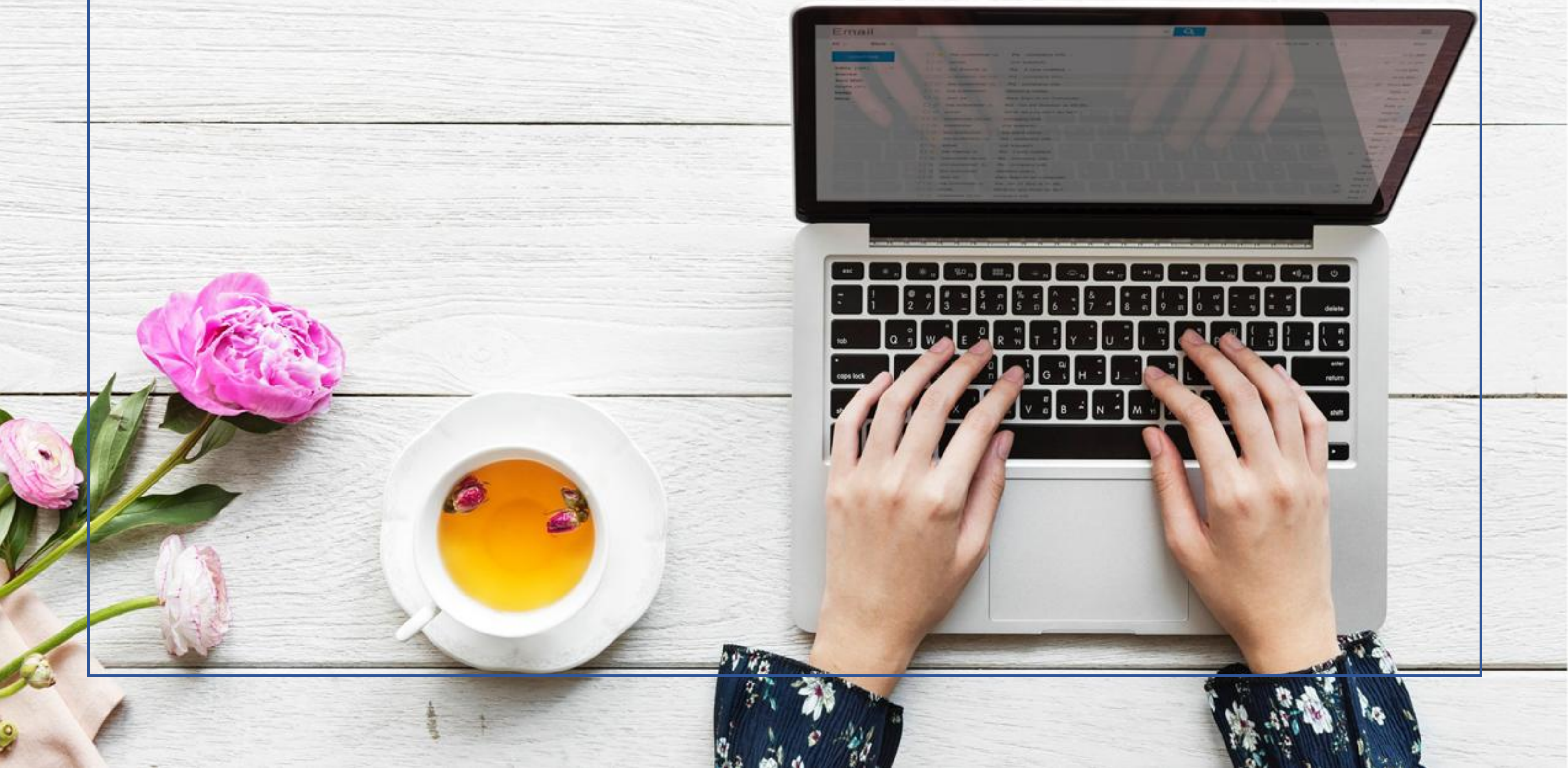
Turning Bazar.com into Amazon: Replication, Caching, and Consistency

Lina Qurom

11924435



Overall Program Design:

Bazar.com aims to enhance its online store to handle increased workload efficiently. The project focuses on implementing replication, caching, and consistency in the existing multi-tier web design with microservices.

Part 1: Replication and Caching

In-memory Cache:

- Implemented an in-memory cache in the front-end server to store recent query results.
- Cache stores book details (id, stock, and cost) for faster retrieval.
- Utilized internal function calls for cache operations, ensuring seamless integration.

Replication:

- Replicated both order and catalog servers on two servers each, as like two machines.
- Implemented a load balancing algorithm to distribute requests among replicas (round-robin).

Cache Consistency:

- Employed a server-push technique for cache consistency.
- Backend replicas send invalidate requests to the cache before making writes, ensuring strong consistency.
- Implemented features like limiting cache size and LRU replacement policy for efficient caching.

How It Works

Query Operation:

- Front-end checks the in-memory cache before forwarding requests to the catalog server.
- Cached results expedite read operations, improving response time.

Buy Operation:

- Write operations (orders) bypass the cache and are processed by the order or catalog servers.

Load Balancing:

- Incoming requests are distributed among replicas using a load balancing algorithm.

Cache Consistency:

- Invalidate requests ensure cache consistency with backend replicas, preventing stale data.

Design Tradeoffs

Cache Implementation:

Tradeoff: Integrated vs. Separate Cache Component.

Justification: Chose integration for simplicity but can opt for separation for scalability in future iterations.

Load Balancing Algorithm:

Tradeoff: Round-robin vs. Least-loaded.

Justification: Chose round-robin for simplicity; future enhancements can explore dynamic load balancing.

Cache Limit and Replacement Policy:

Tradeoff: Fixed vs. Dynamic (LRU).

Justification: Implemented LRU for efficiency; adaptability for different policies in future versions.

Performance Evaluation

- **Average Response Time:**
  - Measured response time with and without caching.
  - Demonstrated the substantial improvement in response time with caching.
- **Cache Consistency Overhead:**
  - Conducted experiments to measure the overhead of cache consistency operations.
  - Analyzed the latency of subsequent requests after a cache miss.

How to Run the Program

Dependencies:

- Python

- Flask
- Docker (optional): I would have liked to implement this part, but unfortunately, time constraints and being alone working prevented it.

Running the Program:

- ➢ Running the Front-end Server
  - python frontend.py
- ➢ Running the Catalog Replicas Server
  - python catalog.py
  - python catalog_replica.py
- ➢ Running the Order Replicas Server
  - python order.py
  - python order_replica.py

  Running all above files each of one in specific terminal, then using curl commands that used in part 1.

Possible Improvements

- ✓ **Internationalization**:

  Enhance support for multiple languages and regions.

- ✓ **Dynamic Load Balancing:**

  Implement a dynamic load balancing algorithm for better resource utilization.

- ✓ **Database Transition:**

  Transition from CSV to a more scalable database solution.

- ✓ **Automated Testing:**

  Implement automated testing for comprehensive validation.

Conclusion

The enhanced Bazar.com demonstrates improved response times, efficient caching, and robust consistency mechanisms. Design tradeoffs prioritize simplicity for the current version, with scalability and feature enhancements considered for future iterations.