FACULTY OF TECHNICAL SCIENCES

DEVELOPMENT OF ELECTROENERGETICAL SYSTEMS

PROJECT – CACHE MEMORY

Team members:

Aleksa Bajat PR78/2019

Dragan Stančević PR84/2019

Anja Puškaš PR68/2019

Jovan Peškanov PR72/2019

Mentor: Zorana Babić

Contents

Idea Behind Cache Memory

Cache Memory is a system implemented using a micro-service architecture. The back-end part of the system is made out of 4 components Writer, Dump Buffer, Historical, and Writer. There is also one more component that is essentially the initiator of the service work – Client.

Writer

Writer is a part of the system with a transient role. When it receives data its only purpose is to dispatch it to the next component. In a more advanced scenario with more time this component could very well serve the role of the load balancer.

Dump Buffer

Dump buffer as its name annotates serves the duty of receiving and containing multiple pieces of data. Data is contained inside a queue data structure. Clarification wise queue is a FIFO (first in first out) implementation which is perfect in order to give priority to data based on the time of retrieval.

Reader

Reader is the intermediary component that propagates all the reading requests from the client towards the part of the system which is directly connected to the database (Historical). Also, it has the role of receiving data from the before-mentioned service and providing it to the client.

Historical

Historical contrary to its name is not doing the job of the conventional logger (monitoring of the entire or part of the system). Instead, it communicates with the database in order to write or read from it based on the service that is talking to it.

Client

Last but not least the client component is given to the end-user with a user-friendly (as much as the terminal can provide) interface. It contains a menu so that the user can pick actions according to their needs.

Why micro-services?

One of the foundational ideas while developing this project was having as much horizontal scaling and loose coupling as possible. This way, every component has its codebase. This decision boosted the team's productivity since the tasks were able to be split up in a way that would generally make the job of merging all the code very easy, with little or no conflicts. Another benefit of this architecture is that debugging is simplified since components have nothing in common apart from delegating the data. Since these micro-services rely on the physical address of another service there is tight location coupling. This could be solved by implementing a messaging system using a common bus and implementing a saga design pattern (manages data consistency across micro-services in distributed transaction scenarios). There is also some form of fault isolation, if one of the components goes to a faulty state others will continue to work. Testing new technologies and implementations is very convenient since it requires changes in just one component, unlike a monolithic system where this kind of change would cause a large portion of code to break. If this project were to be continued and if it were to grow this kind of development would let entire teams form based on certain functionalities of the system and that again provides loose coupling even in real-life organization.

Technologies

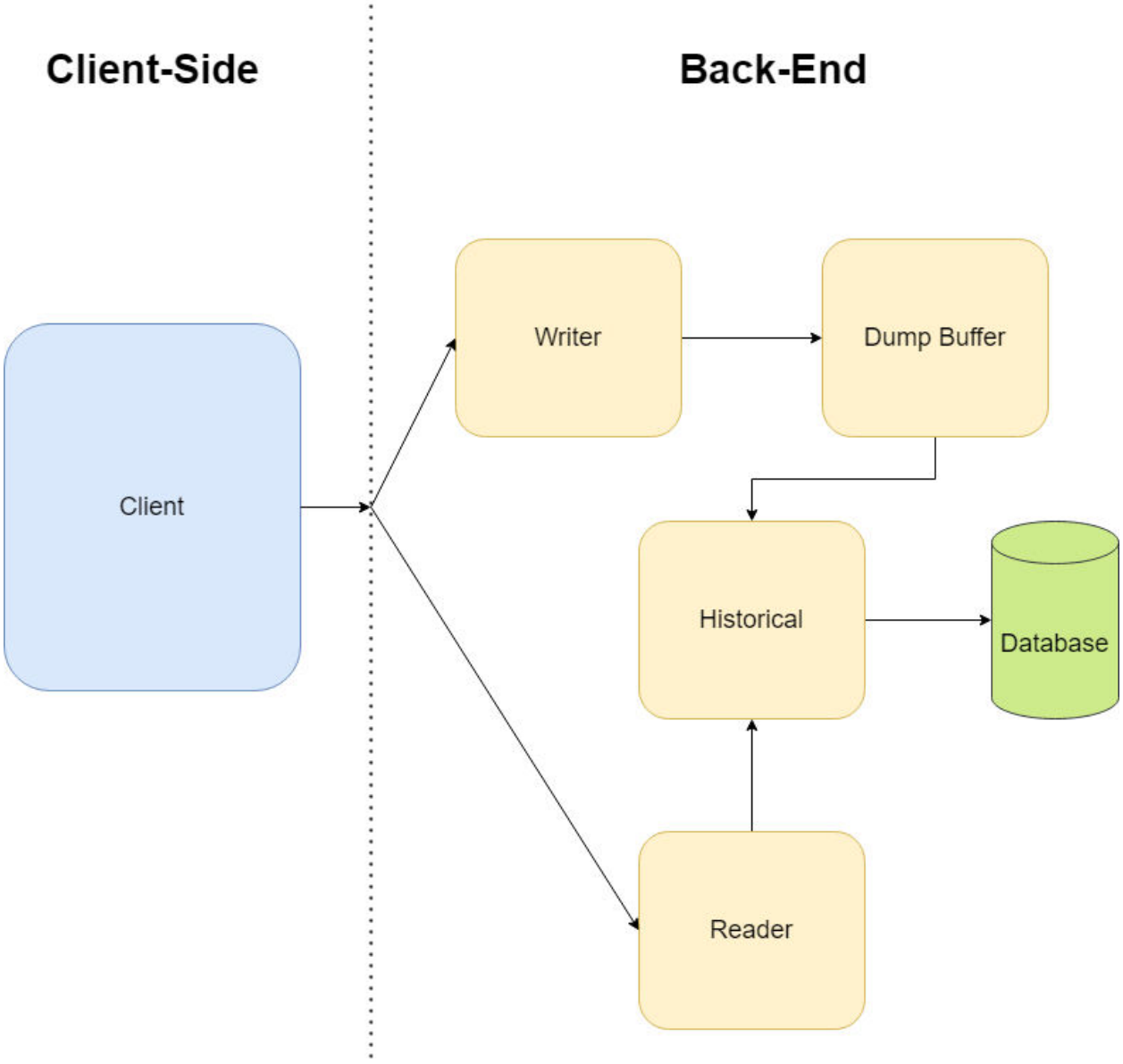
The programming language used to realize this project was Python 3. To support a good directory management Sys module was used to change the root path so that interpreter can see common modules or in case of testing all of them. The socket module was there to support TCP communication between components. The team opted for TCP instead of UDP because reliable data transport was a necessity with given specifications. There was minimal use for the Time module since the time of submittal to the database is important. The dump Buffer component relies on the Queue module to realize chronological data dispatching. Even though some would deem it's much unsafer than JSON, the Pickle module was used heavily to support the transport of nested classes (generally all the data). Module _thread is also a key to making everything work together, especially when having distinct but necessary tasks (wouldn't make sense to make even more services because we would produce a anti-pattern – dependency maps) for a micro-service to work. Last but not least, sqlite3 because of its close integration with Python.

Usage of Ports

Table 1: Ports

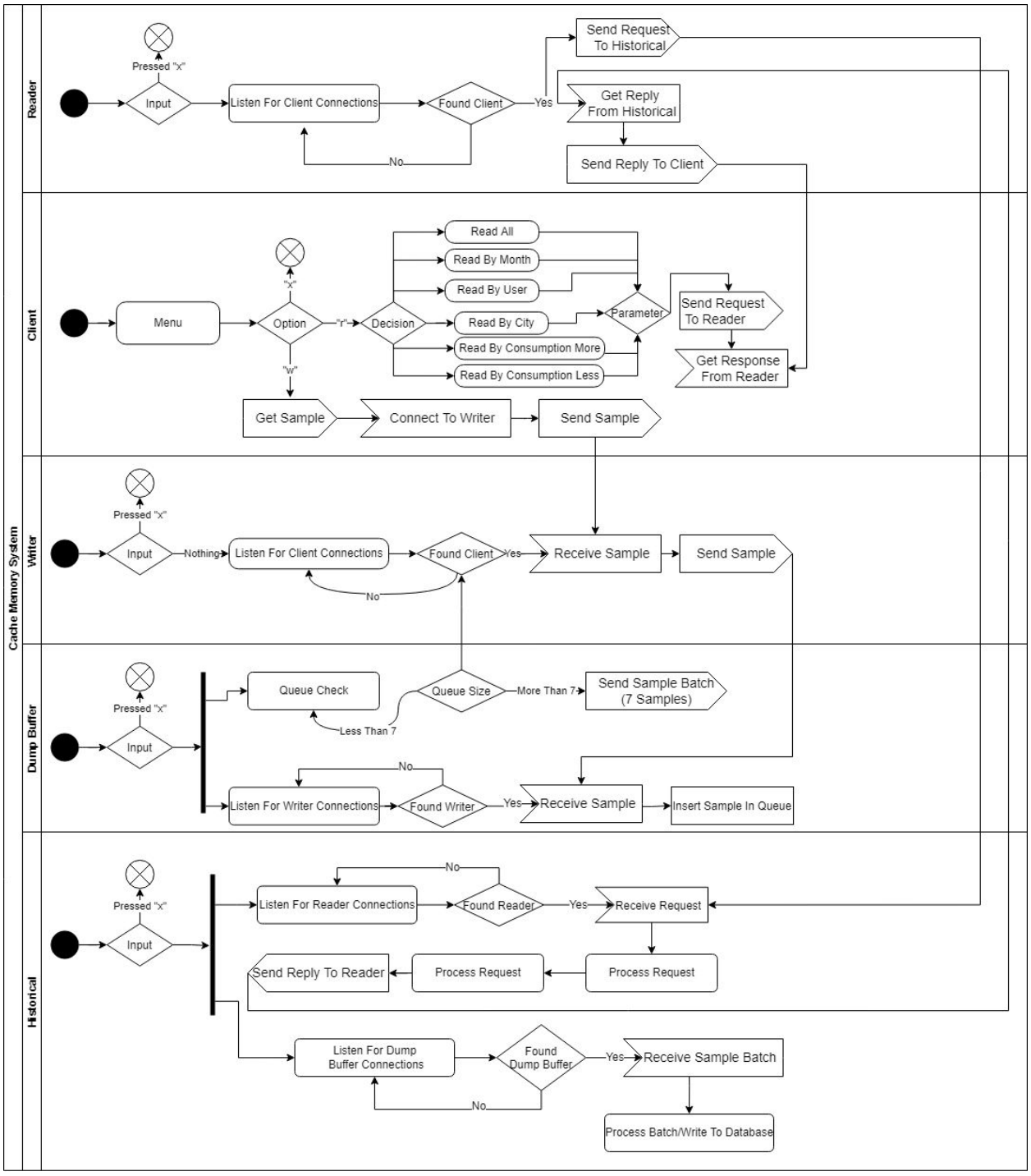
Ports Used In Communication		
Micro-Service	Listen At	Send To
Client	/	10000, 40000
Writer	10000	20000
Dump Buffer	20000	30000
Reader	40000	60000
Historical	30000,60000	/

High-level Perspective of the System



System Overview

Activity Diagram



System Overview 2