Distributed High-Frequency Trading System[1]

Tani Diament Yeshiva College Computer Science

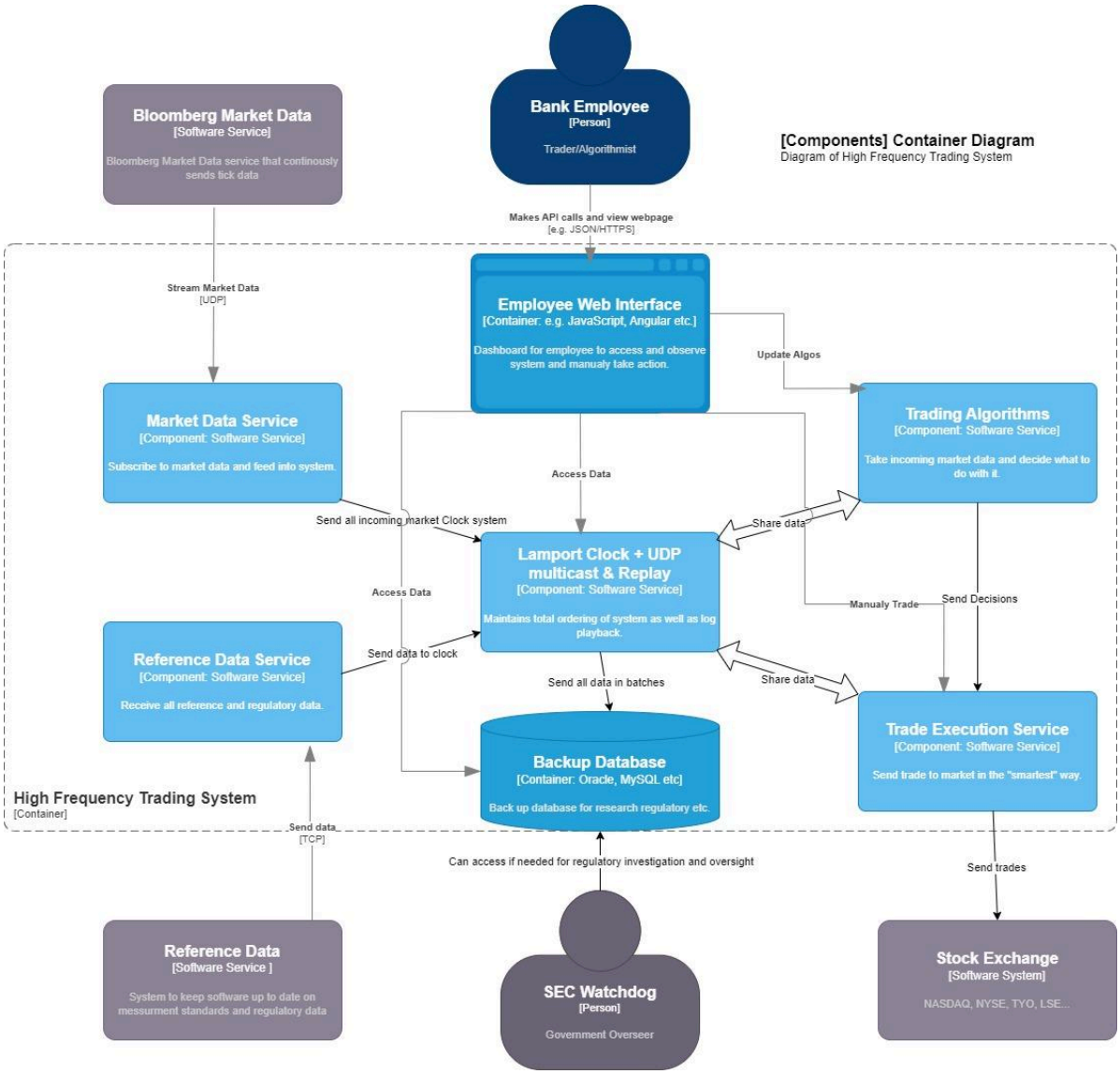

Figure 1: High level overview of the system.

Abstract

The High-Frequency Trading System(HFTS) is designed to make split second decisions based on current market data and conditions to optimize profit. To gain a competitive advantage we need to be able to quickly scale our system based on market volatility and trading volume. These quickly changing market conditions present challenges for scalability, reliability and availability of a HFT system. This paper presents a possible solution to this modern day challenge.

[1] Created for the Advanced Distributed Systems Course (com3810) semester project at yeshiva university. Designed under the guidance of Proffesor Judah Diament. Inspired by a guest lecture from Breman Thuraisingham and Cheng Niu on distributed systems at Nomura.
Link to GitHub

1. Introduction

Our HFTS needs to import data from the market, in our case streamed from Bloomberg[2]. It must have the ability to execute trades in markets around the globe (NYSE, TYO, LSE…). It requires a dashboard for bank employees to view real time data and the ability to update algorithms and manually place trades. The system also needs to have kill switches and controls that keep the system from spiraling out of control in case of errors. All of this has to happen on a distributed computing platform with ability to dynamically scale and be fault tolerant. Given that speed is essential for the success of the system much thought has gone into the networking architecture.

2. Data Services

The backbone of the HFTS are the data services that provide the necessary data for the system to run. These services must have high availability and reliability. We also need a service to gather market data and reference data. We also need a service that can store data long term for compliance, research or testing purposes.

2.1 Market Data Service

Our Market Data Service(MDS) relies on data streamed from Bloomberg over UDP. The UDP protocol was chosen for streaming this data as we need a very high speed communication system because of the large amounts of data that flow into the system. It is okay if the system does not receive some messages. The MDS formats the incoming data into the system specific event object. These objects are then streamed via UDP to the Lamport Service (described later in paper) which then sends it to the components that need the data. (See figure 2)

[2]All references to Bloomberg are theoretical and do not necessarily represent real Bloomberg services.

One of the challenges for this service is the possibility of the server hosting the service being a single point of failure and would bring the system to a halt if it failed. If such a failure occurs it can lead to financial loss and will require a system admin to configure a new server to take up the role of MDS. To solve this problem we create a cluster(size configurable by sysadmin) of identical servers that are on standby in case the active server goes down. The servers send heartbeats every two seconds to every other server in the cluster. Once the new server is considered dead the LCS will no longer accept messages from the old MDS server (this is true for the RDS as well and the leaders of the alg and execution services).

If five seconds elapse without hearing from the active MDS, the server is declared dead. The next server is chosen based on a ZooKeeper style leader election where the deciding factor in the vote is server ID. The winning server sends a message to the Lamport Service to mark the previous MDS server as dead and to ignore all messages that may come from it, a dead MDS server cannot be resurrected. The new server then takes the role of the MDS.

2.2 Reference Data Service

Our Reference Data Service(RDS) has a similar architecture to the MDS with some key differences. All the data is streamed from Bloomberg but in contrast to the MDS we receive the data over TCP. This protocol was chosen as the missing a message regarding reference data is unacceptable in the system. The reference data includes regulatory information as well as other measurements that are crucial for system performance as well as regulatory compliance. The RDS formats the data into a system wide event object and is sent via UDP to the Lamport service. We are not worried about message loss over UDP internally given the

stable physical connection between the services. The RDS uses the same cluster and failure procedures as the MDS. (See figure 3)

2.3 Backup Database Service

One of the core parts of the HFTS is the Backup Database Service(BDBS). The BDBS's purpose is to store all the system's data for later access by users. This service is never used during the regular algorithmic trading flow unlike the rest of the services in the HFTS.

The BDBS is built upon a standard relational database of the sysadmin's choosing. This database can easily be sharded for scale and replicated for reliability as data loss is not tolerated. This is configured at launch of the system.

The BDBS has two main processes, the first is the data input process and the second is the database's API. The database input process receives data in batches from the Lamport Service, this data is in the system wide event format. The process then reformats the data for the relational database and inputs it using SQL. The second process is the API which can be queried via the Employee Web Interface. This process gets the requested data from the database via SQL then transforms it to the system wide event object and returns the data to the requester. (See figure 4)

3. Algorithmic trading

The heart of the HFTS system are the algorithms and the trading execution services. These services require the most compute and therefore will be scaled more greatly than the other services. The complexity and speed of these systems can easily lead to catastrophic failures so they must be designed with much caution. All inter system communication is done over UDP for speed, it is okay for a piece of data or two to be lost, especially since that will happen very infrequently. (See figure 5)

3.1 Trading Algorithm Service

The Trading Algorithm Service(TAS) uses mathematical models to make decisions on current market data to make trades. These mathematical models are binaries and are imputed by the sysadmin at boot and can be changed via the Employee Web Interface. The market and reference data is streamed from the Lamport Service which establishes a global time for the system as we will discuss later. Once the algorithm makes a decision it sends the decision to the Trade Execution Service which we will discuss later. (See figure 6)

One of the dangers encountered in a system like this is for an error in the system to cause trades to execute in a manner that will either lose the bank money or not conform with SEC and other government regulations. This can happen if a bug makes the service execute trades that are not profitable or illegal and given the speed of the system that can spiral out of control very quickly. To combat this the sysadmin enters a set of controls and kill switches that limit the system and prevent the system from spiraling out of control. These controls are inputted as a binary into the system.

Given the fact that this service is compute intensive it is necessary to distribute the load to multiple servers. One challenge with this is that we cannot simply do a round robin scheduler that sends each incoming event to the next server as the algorithms base their decisions off previous data from that stock. To solve this challenge we divide the load based on the ticker of the stock so the same server will always be dealing with the same set of stocks. The system divides the stock based on previous volume data and these weights are used to divide the stocks up among the available servers. The system can redevide the stocks if one server sees more load because of volatility in one or more of its stocks. A sys admin can also add a server to join the cluster if we want to process more data.

The system is architected with a master server that gets all the incoming data from the Lamport Service and then sends it to the corresponding worker that is in charge of that ticker. When the worker finishes calculating if it decides to make a trade it sends it to the Execution Service. The servers use the same heartbeat method as the MDS and RDS and when the leader goes down a new server is elected leader via Zookeeper Leader Election where the server experiencing the highest load is elected leader (tie broken by server ID). The reason we choose the server with the current highest load is because when the new leader takes on the role of leader it distributes the stocks and corresponding data to the remaining worker servers so we want to make sure a high load server does not cause a failure. After the election the new leader requests the Lamport service to replay whatever the TAS missed since the leader went down. The new leader queries the workers to find out what the last event was that the dead leader received before it went down and the new leader continues from there. In the case of a worker failure we divide its stocks among the remaining workers and we send them the backlog of relevant market data from the Lamport service.

All decisions that are made by the TAS are sent to the Lamport System in the event format to establish a global ordering of events. This also helps us in the case of worker failure so we know what trades the now dead worker already sent to the Execution Service.

3.2 Trading Execution Service

The Trade Execution Service(TES) is responsible for executing the trades from the TAS and the bank employees. The TES has two parts, (1) it must decide which exchange is the best to execute the order on and how fast or slow to execute to reduce market slip and to hide out strategy from our competitors we call this the Smart Order Router. (2) It must also execute the order on the given exchange that was chosen in step one we call this the Order Gateway. The TES is limited by the same controls and kill switches that the TAS is limited by. (See figure 7)

The TES uses the same distributed model as the TAS with a master sending the order to the corresponding server in charge of that stock. The worker then calculates the best exchange to execute the trade on. Once it completes the trade it sends the information to the Lamport System in the system wide event format.

4. Lamport Clock and System Time

One of the challenges that any distributed system faces is how to establish global ordering. This challenge is especially important in a HFTS where all decisions are based on what happened when. We also need a very reliable system because if it fails the system will grind to a halt (see 4.4). The Lamport Clock Service(LCS) has three main components (1) the clock itself that keeps global system time (2) the UDP multicast service that streams the events to all components of the HFTS (3) the replay service for components that have either just booted or were down for a period of time so they need to catch up to the current system state. These three services run on the same server on different cores. (See figure 8)

4.1 Lamport Clock Service

The main component of the LCS is the clock itself. All events that take place on any component of the HFTS are streamed to the LCS which timestamps them with a long starting from 1. The LCS immediately writes it to a queue and an in memory log file which will be used by the UDP multicast and Replay services respectively. The LCS is responsible for sending the most recent log data to the BDBS. (See figure 8.1)

4.2 UDP Multicast Service

The UDP Multicast Service(UMS) is responsible for streaming all the events that come into the LCS. The UMS reads the events off the queue as they come in and it streams them to all the system components over UDP. This ensures that all the services have the same event ordering.

4.3 Replay Service

The Replay Service(RS) is a service that can be contacted in the case where a system component missed out on the multicasted data from the UMS. If a service is rebooted, has a failure and election, or a temporary glitch it sends a message to the RS with the system timestamp of the last event it received. The RS then sends the data to the requester over UDP and catches it up to current system time. Once the service is in sync with the LCS the RS stops sending more data. This service also is used by the Employee Web Interface when it boots to show real time data. (See figure 8.2)

4.4 LCS Reliability

Like many components of the HFTS we need the LCS to be very reliable. To accomplish this we take a similar approach to that of the MDS. The LCS is made up of a cluster of 3 servers with one serving as the leader. Any message that is sent to the leader is forwarded to all the followers. Each follower keeps its own copy of the log locally on its server. In the case of leader failure which is determined by lack of heartbeat (event messages from the UMS) for a given time a new leader is chosen with Zookeeper based on the server ID. When the leader goes down the TAS and TES will automatically stop after a few seconds as they will stop receiving data but the data services will not. To solve this when the new leader comes up it sends a message one by one to all the services to catch up from the last event it has. The services all know the global ordering of the events because they receive the logs from the UMS so they know where to start from. Because of this failure scenario all services keep a five minute micro log in case they have to catch the leader up. A sysadmin can add a new server to the cluster in case of a leader failure.

5. Employee Web Interface

The Employee Web Interface(EWI) is the portal for bank employees to interact with the HFTS. It runs an HTTP server for the client side website to interact with the system. The EWI allows for changing of the controls and algos, viewing current system data and data from the BDBS, and allows for manual trading. Given the fact that the EWI is mainly used for data consumption it is run on one of the cores of the LCS server. The LCS keeps a map of services to address so the EWI can get the address of a service it wants to interact with e.g. to execute a trade. When changing the Algorithms or controls that is done via an Event sent to the LCS which will then be sent to all the services. This service also allows for data access for governing bodies like the SEC. (See figures 9&10)

6. System Boot

The first step in system boot is the configuration of the LCS as that is the “heart” of the system. The LCS is booted with three servers given their address, replica addresses and the address of all the other components. We then boot all the other machines via their Java class with their address, their peers and the address of the LCS along with any component specific details like the address of the exchange for the TES.

As soon as run() is called on all the systems everything will automatically be up and running. Every system with more than one server will start with an election to find a leader for that service. That leader will then send a

message to the LCS with the address of the cluster leader for that service. All services only have one server executing at a time and start with three servers besides the TAS and TES which divide the work. The TAS and TES can be started with any amount of servers depending on how much data the system will be trading on. A cluster of three is minimum for startup but more are recommended when dealing with larger markets.

If any address is updated amongst the services it will notify the LCS which will then notify the other services via broadcast as it is a standard event. All in system communication is done via UDP in the Event format.

7. Performance

Given that performance and speed is central to the HFTS there are some things we can do to ensure high performance. One method is pinning processes to the cpu so that they will not be context switched which can take valuable time. Even though each service is on its own physical service a standard configuration on a JVM or OS will still context switch the main process with others. To make sure we do not take a hit to performance we pin the main process(es) to the cpu core.

Diagrams

Figure 2: The market data service streams market data from Bloomberg to the LCS.

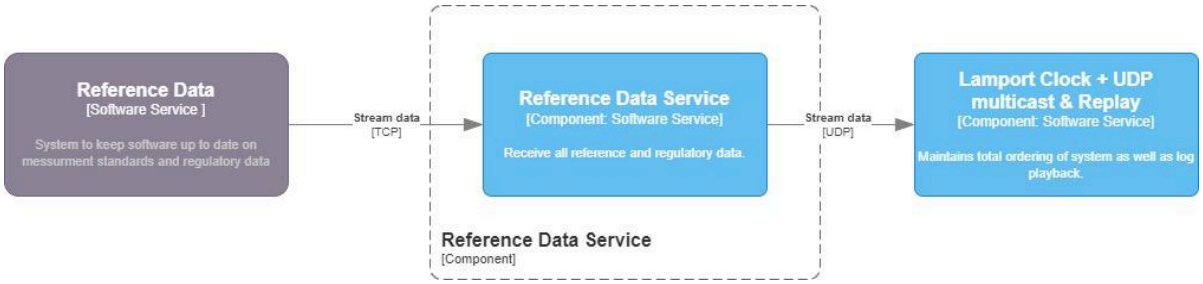

Figure 3: The reference data service streams reference data from Bloomberg to the LCS.

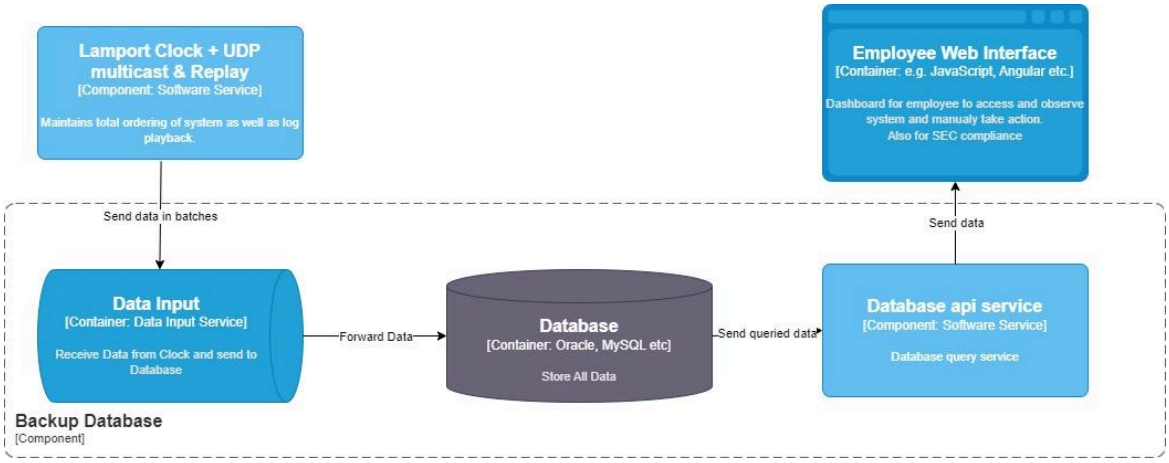

Figure 4: The BDBS stores all system log data, and has an API for queries.

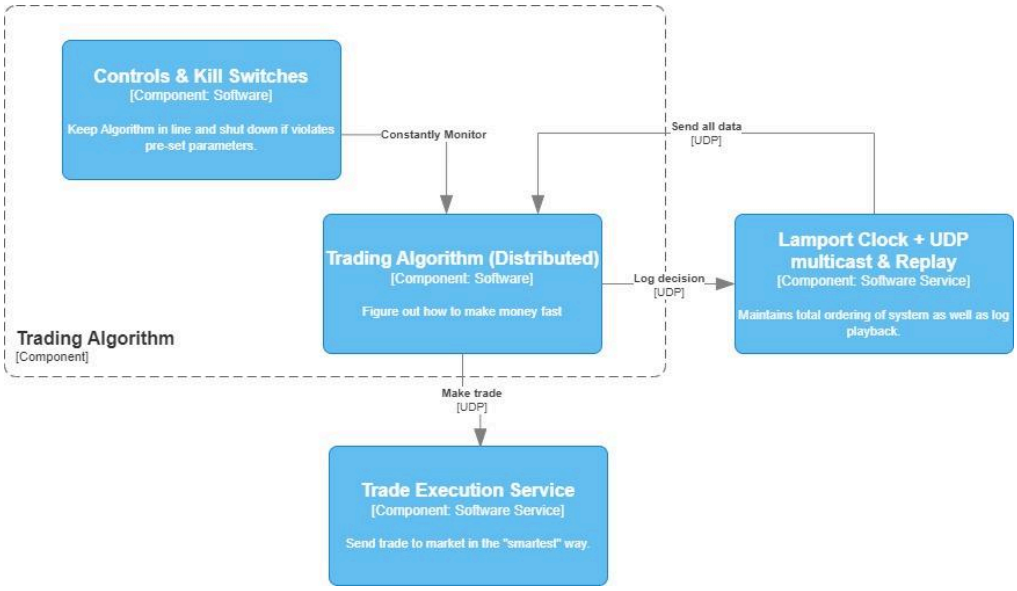

Figure 5: Logical high level diagram of trading flow.

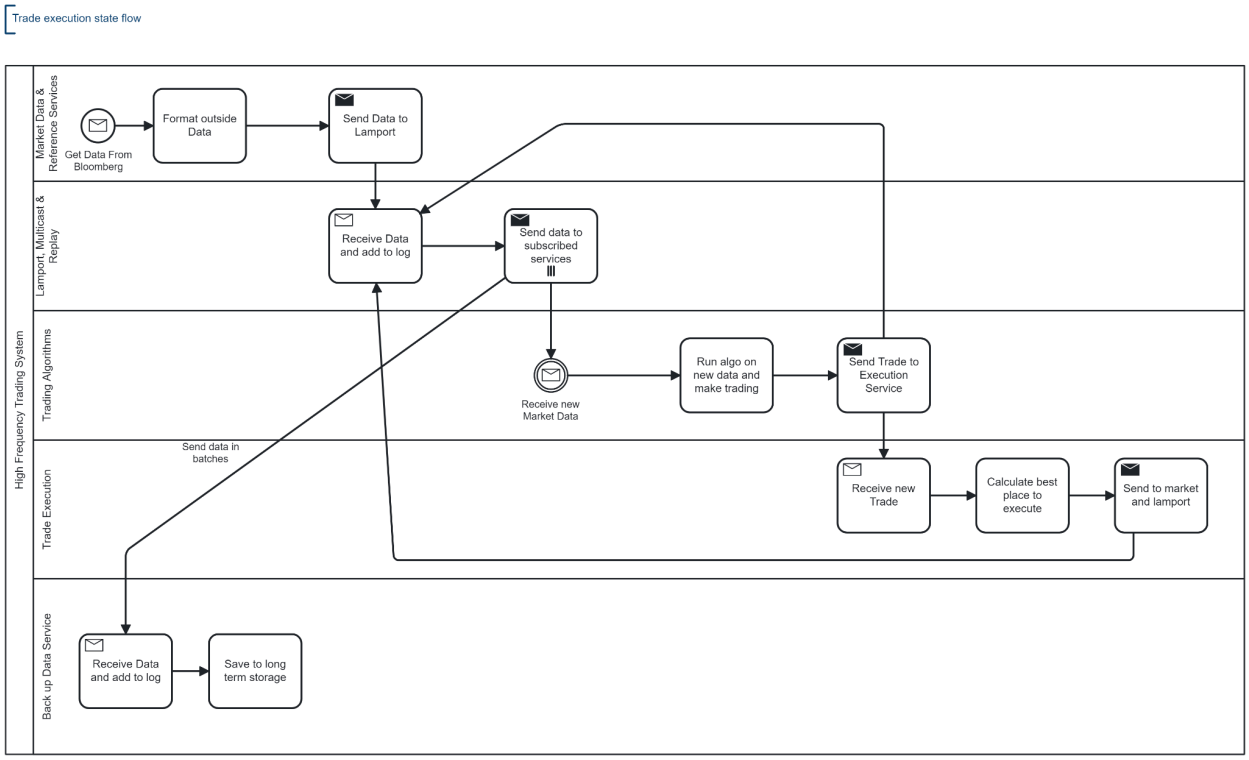

Figure 6: Trade execution flow

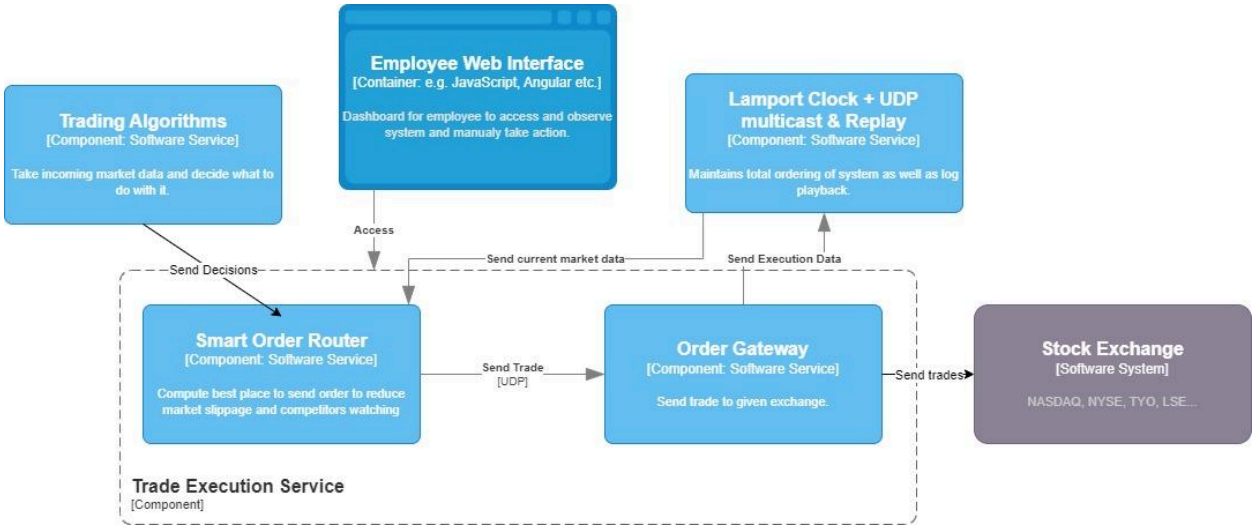

Figure 7: Trade Execution Service

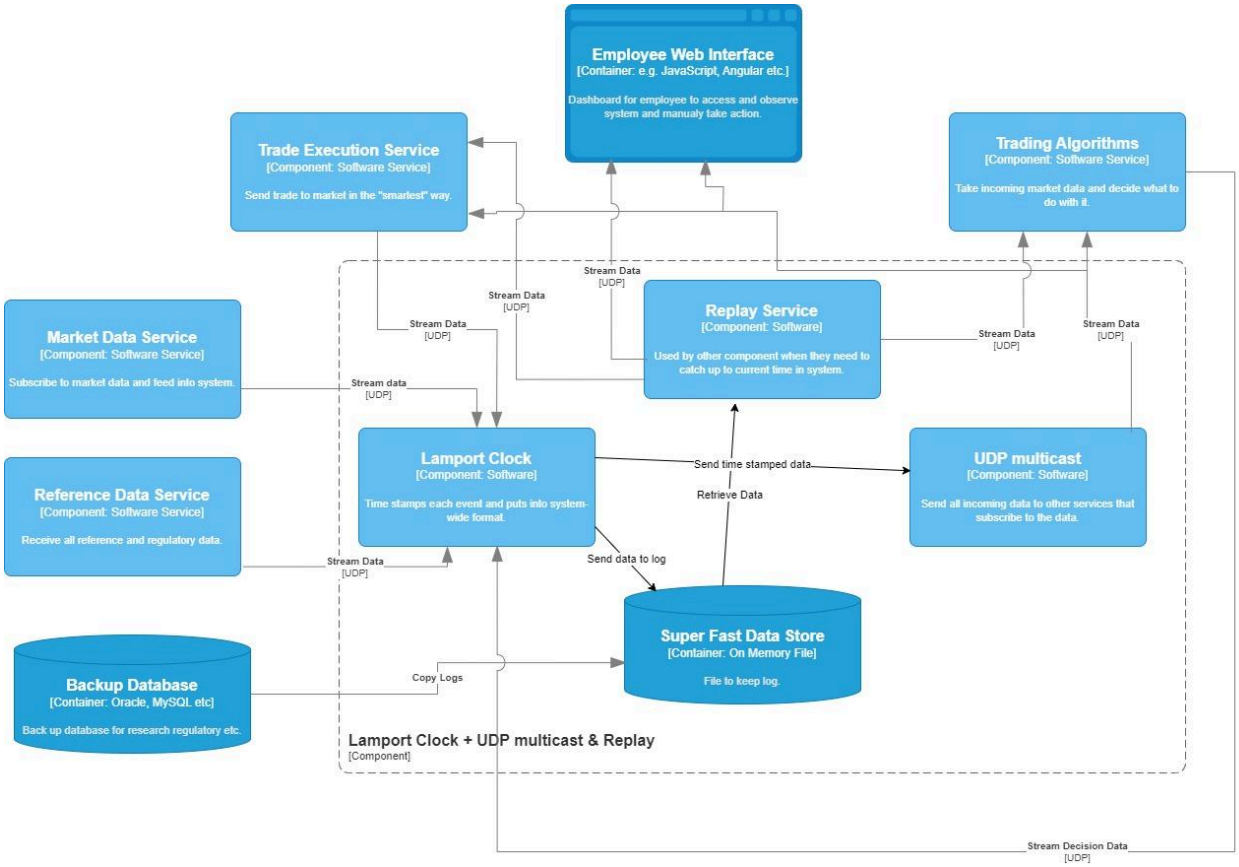

Figure 8: Lamport Clock Service.

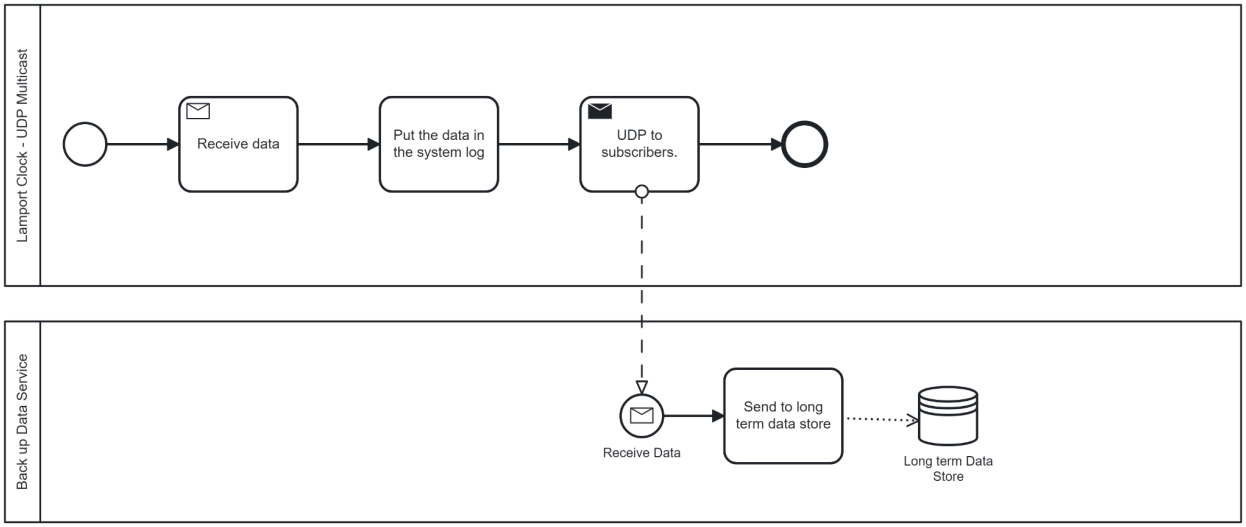

Figure 8.1: LCS flow.

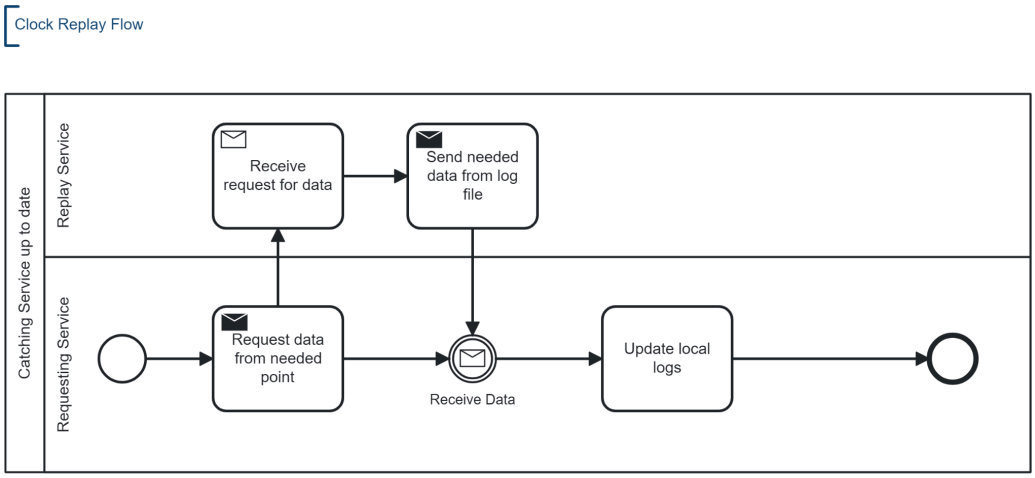

Figure 8.2: Replay Service flow.

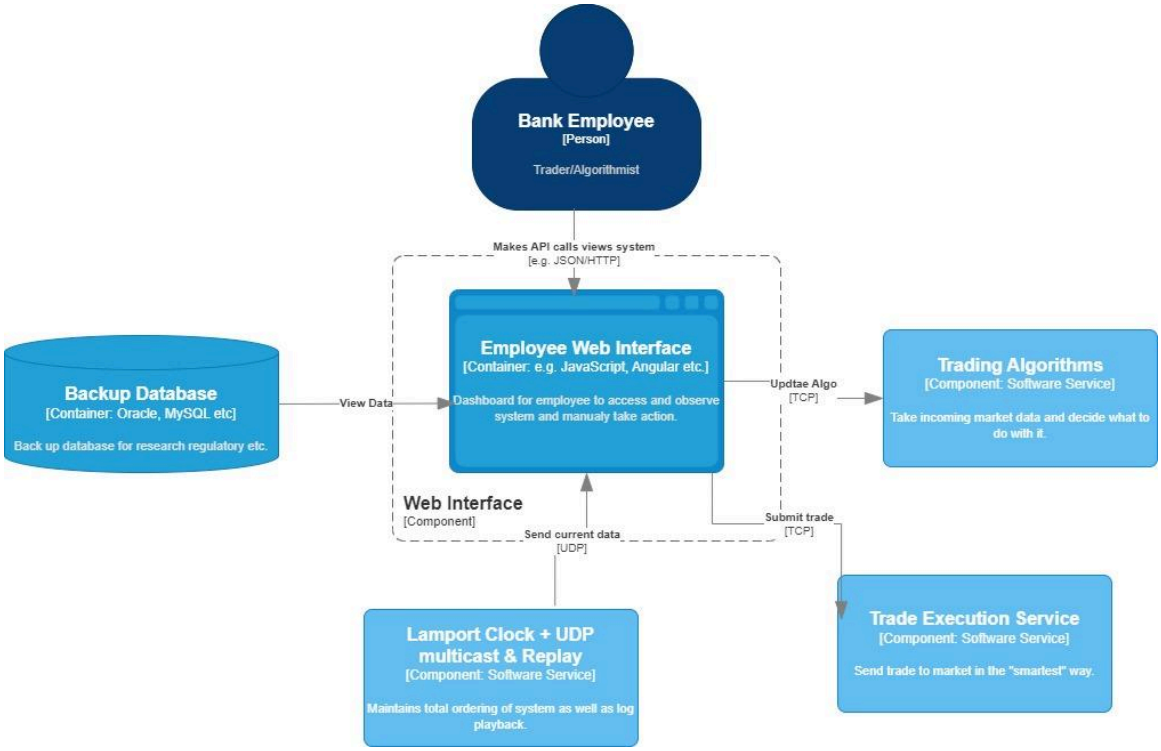

Figure 9: Employee Web Interface

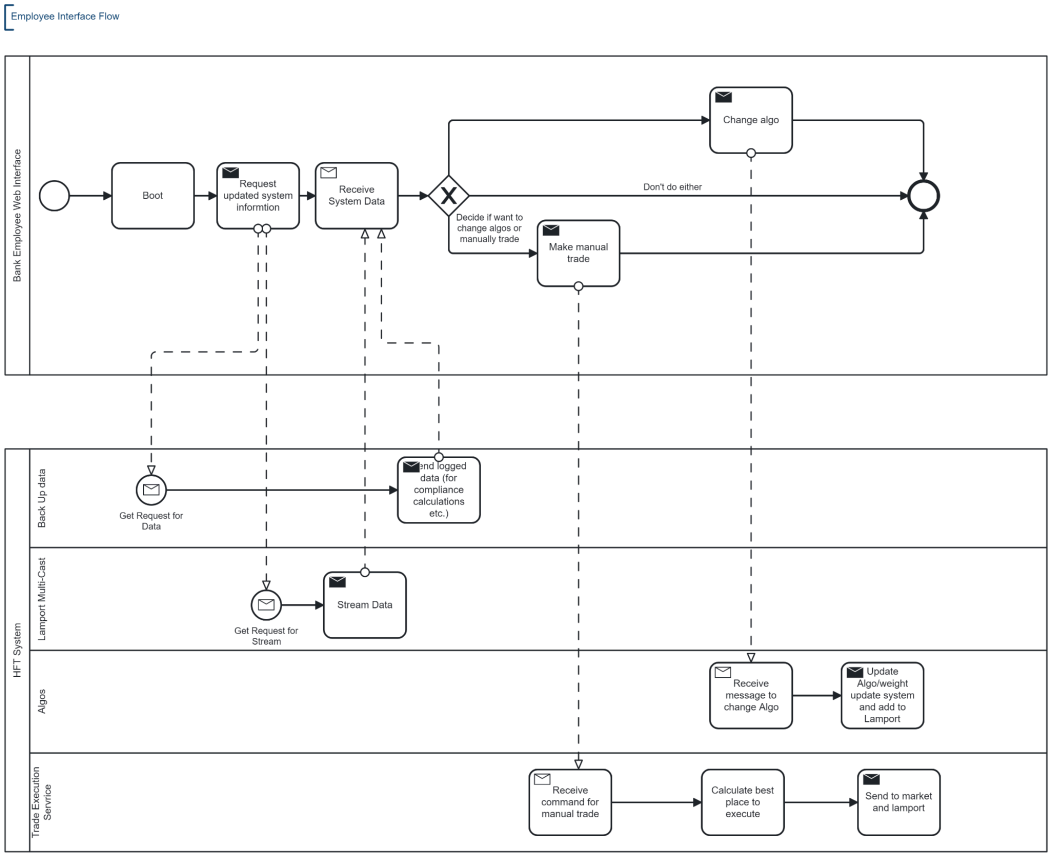

Figure 10: Employee Interaction Flow