Central Bank Digital Currency Experiments

Results and Findings from "Proof of Concept Phase 2"

Payment and Settlement Systems Department, Bank of Japan

May 2023

Contents

1 Objectives 1

2 Scope of experiment 2

3 Main results 4

3.1 Additional functions 4

3.1.1 Economic design/Improving convenience of payments 4

3.1.2 Coordination between intermediaries and connection with external systems 11

3.1.3 Issues in the possible event of social implementation 17

3.2 Potential use of new technologies 19

3.2.1 Flexible-value token model 19

3.2.2 NoSQL Databases 22

4 Conclusion 24

Appendix 1: Alternative system architectures 25

Appendix 2: Details of the performance evaluation test 28

1 Objectives

The Payment and Settlement Systems Department of the Bank of Japan (BOJ) conducted its "Proof of Concept (PoC) Phase 1" from April 2021 to March 2022, building an experimental environment using several design alternatives for a Central Bank Digital Currency (CBDC) ledger as the foundation of a CBDC system, to evaluate whether its basic functions could be processed appropriately.[1]

In PoC Phase 2, conducted from April 2022 to March 2023, the BOJ evaluated additional functions regarding which it would be desirable to confirm technical issues as early as possible, by adding them to the basic functions of the CBDC ledger, built in Phase 1. The results were evaluated in terms of processing performance and technical feasibility. Further, the potential use of new technologies that were not included in Phase 1 was explored with respect to data models and databases.

Whether to issue a CBDC should be decided by discussions among the Japanese public. With a view to facilitating such discussions, the BOJ will continue to make such thorough preparations as conducting technical experiments to respond to changes in circumstances in an appropriate manner.

[1] For the results of the PoC Phase 1, see the following reference: Payment and Settlement Systems Department, BOJ, "Central Bank Digital Currency Experiments Results and Findings from 'Proof of Concept Phase 1'" (May 2022).

2 Scope of experiment

In Phase 2, additional functions of CBDC were evaluated in three separate blocks. Although the assumption is not that these functions would be adopted in the possible event of social implementation, it is desirable to identify technical issues early on so as to contribute to future decision making (see Figure 1).

Figure 1: Additional functions in Phase 2

Economic design (Safeguards ensuring the stability of the financial system)	● Limits on holdings ● Limits on amounts and number of transactions ● Swing function (automatic conversion of amount in excess of holding limits into bank deposits, etc. or automatic conversion of remittance receipt into bank deposits, etc. based on user attributes) ● Application of interest on holdings
Improving convenience of payments	● Scheduled remittance instructions by users ● Batch remittance and pull payment at user request
Coordination among intermediaries/ Connection with external systems	● Providing multiple accounts to one user ● Limits per user on holding and transaction amount/number, based on the above assumption ● Methods for connection with external systems

Additional functions of the economic design block cover various types of safeguards for preempting a sudden shift from bank deposits to CBDC. Further, the functions of improving convenience of payments block cover aspects that could lead to improved user convenience in payments. Regarding the functions of coordination among intermediaries and connection with external systems block, the methods necessary for providing multiple accounts to one user and for connection with existing external systems were evaluated.

Further, the potential use of new technologies that had not been evaluated in Phase 1, such as a flexible-value token model and NoSQL (Not only SQL) databases, was also evaluated (see Figure 2).

Figure 2: New technologies in Phase 2

Flexible-value token model	● Phase 1 evaluated the fixed-value token model in which the token's face value is fixed and tokens are converted to ones of lower value as needed ● Phase 2 explored the flexible-value token model in which the token's face value changes as tokens are merged or split as needed
NoSQL database	● Use traditional relational databases (RDB) from phase1 ● Phase 2 explored the potential for using NoSQL (Not only SQL) databases, which are becoming increasingly popular as databases are not RDBs

Although a token-based data model can be interpreted in various ways depending on the context, in the PoCs, the model is one in which a unique identifier (ID) is assigned to monetary data with a face value, and the CBDC holding status can be recognized by linking this ID with a user ID. In Phase

1, we evaluated a fixed-value token model in which the face value is fixed like cash and the user to whom the existing tokens are linked is changed after the tokens are converted to ones of lower value as necessary at the time of remittance. In Phase 2, we explored a different token-based data model -- the flexible-value token model -- in which existing tokens are merged or split as necessary at the time of remittance and users are linked to newly created tokens.[2]

Additionally, we used a traditional relational database (RDB) as the system database in the experimental environment from Phase 1 onward. However, because a CBDC system requires extremely high processing performance, in Phase 2, NoSQL -- databases other than those based on RDB, some of which are said to have high processing speed and performance scalability -- were explored for possible use.

The ledgers evaluated in Phase 2 were comprised of the following four designs: (i) (a) centralized management by the central bank or (b) shared management between the central bank and intermediaries for the system architecture and (ii) (a) account-based or (b) token-based for the data model. In Phase 1, Designs 1–3 were evaluated, and in Phase 2, Design 4 was additionally evaluated (see Figure 3). Regarding Design 3, while the fixed-value token model was evaluated in Phase 1, the flexible-value token model was evaluated in Phase 2.

Figure 3: Ledger designs in Phase 2

	Centralized	Shared
Account-based	Design 1	Design 2
Token-based	Design 3	Design 4

= Phase 1 scope

+ = Phase 2 scope

[2] For details on the difference between fixed-value and flexible-value approaches in token-based ledger systems, see Appendix 1 of the "Central Bank Digital Currency Experiments Results and Findings from 'Proof of Concept Phase 1'."

3 Main results

The experiment results of additional functions are described in 3.1 and those of new technologies are described in 3.2.

3.1 Additional functions

3.1.1 Economic design/Improving convenience of payments

The following functions were evaluated for economic design.

(1) Setting upper limits on holdings (holding limits)

(2) Setting upper limits on amounts and number of transactions for each transaction or for a specific period of time (transaction amount/number limits)

(3) Automatic conversion of (a) excess holdings into bank deposits and other forms of private money or (b) remittance receipts into bank deposits and other forms of private money based on user attributes, for example, corporations and individuals (swing function)

(4) Application of interest on holdings (Interest application)

The following functions were evaluated for improving convenience of payments.

(1) Scheduled remittance

(2) Batch execution of multiple remittances (batch remittance)

(3) Execution of a remittance as a payment request initiated by payee (pull payment) [3]

Experimental environment

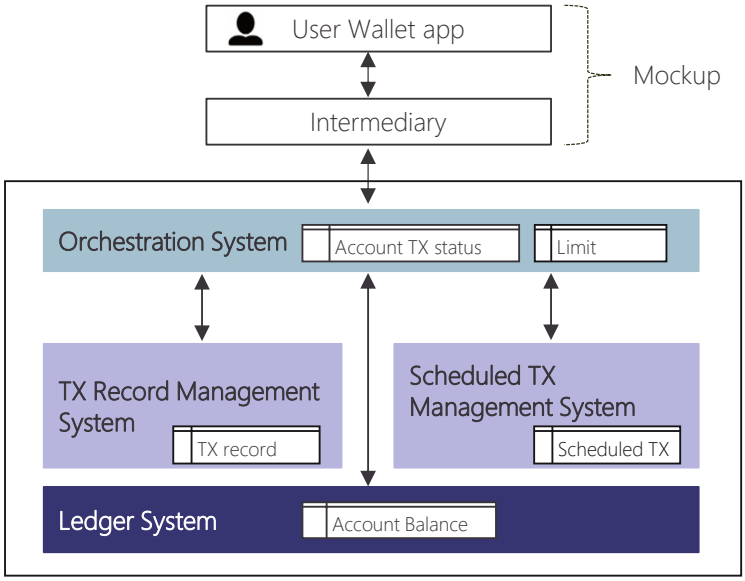
Experimental work was conducted using Design 1, which has the simplest configuration, to ensure that the system could be built on the public cloud and evaluated efficiently within the project's time constraints. For Design 2, which uses the same account-based data model, architecture evaluation was conducted using the results of the performance evaluation test from Design 1.[4]

[3] Direct debits are examples of existing pull payments.

[4] Designs 3 and 4, in which the data model is a flexible-value token model, need to be evaluated in terms of their basic functions; the results are described separately in Section 3.2.

Multiple systems were built and an application programming interface (API) was used to link the systems. Specifically, in addition to the ledger system built in Phase 1, a transaction record management system that references the amount/number of transactions when transaction amount/number limit checking is conducted and a scheduled transaction management system that automatically activates transaction requests by the user were added to the CBDC system. Additionally, an orchestration system[5] was added as a facade to ensure data consistency between systems; it functions as a contact point for processing transactions in order, manages the status of account processing in each system to ensure data consistency between systems, and performs mutual exclusion control so that no other transactions are processed during the execution of a given transaction in principle. Through an orchestration system that maintains a database for managing the upper limits of various restrictions, we also implemented a process of checking whether or not the upper limits have been violated in the system. The ledger system was not configured[6] to have functions such as transaction record management directly on it because it was deemed that the system should not perform any processing other than updating balances as much as possible, considering that it would be subject to a high updating load in the possible event of social implementation.

Figure 4: Experimental environment in Design 1



Note: TX=transaction

[5] In the processing of a certain transaction, the mutual exclusion control is performed so that, in principle, other processing for the same account is placed on standby; it is executed in the following order: transaction limit checking (amount and number), holding limit checking, balance update, and transaction record update. However, to improve performance, a mechanism was adopted to allow other processing to proceed when consistency can be ensured through various limits processing (e.g., when a combination of operations such as payout and remittance where one increases and the other decreases, processing can proceed without having to wait for both operations to be processed).

[6] See Appendix 1 for a discussion of these alternative system architectures.

As in Phase 1, the design of the intermediary system and the user's wallet application was simplified to only submit transaction instructions, and the performance evaluation focused on the system's processing from start to completion after receiving the transaction instructions from the mockup (see Figure 4 for the architecture and Appendix 2 for details of the performance evaluation test including the server specifications used).

Implementation method in an experimental environment

Under the above architecture, each additional function was implemented in the experimental environment as follows.

Various limits

The orchestration system conducts the process while referring to the information in other systems for limit checking (i.e., the CBDC balance managed by the ledger system and the transaction record managed by the transaction record management system). After the checking process, the same remittance processing as the basic function implemented in Phase 1 is performed on the ledger, and the transaction record is updated after the ledger is updated.

Swing

After the orchestration system calculates the amount in excess of the holding limits and identifies user attributes, such as corporate or individual, it sends an automatic acceptance instruction to convert the subject amount to bank deposits, etc., to the ledger and the intermediary mockup.

Interest application

Interest receipt and payment are exempted from the transaction amount/number limits. Therefore, it is set to proceed without checking whether there is a violation of the transaction amount/number limits and is implemented using the scheduled remittance mechanism described below.[7]

Scheduled remittance

The implementation is based on the premise that the remittance instructions to be executed in the future are registered in advance in the scheduled transaction management system. The remittance instructions are sent one by one from the scheduled transaction management

[7] The implementation is based on the premise that, for each account, the amount of interest applied is calculated based on its CBDC balance within a certain period of time and on a predetermined interest rate. Further, a CBDC account at the central bank has been established; it is used to receive and pay CBDC of interest amounts through the user's account (treated as a transfer between accounts in the system). There are alternative methods like directly increasing or decreasing the CBDC balance in user accounts (treated in the system as issuing or redeeming CBDC to each account), without the involvement of the central bank account.

system to the orchestration system at a specified time[8], after which they are processed in the same manner as a standard remittance.

Batch remittance

The orchestration system is set up to obtain all upper limits for multiple remittance instructions at once, then split them one by one, and thereafter process them in the same manner as a standard remittance.

Pull payment

The implementation is based on the assumption that the payee initiates the remittance instruction based on the prior consent of the payer. The subsequent processing is the same as for a standard remittance.

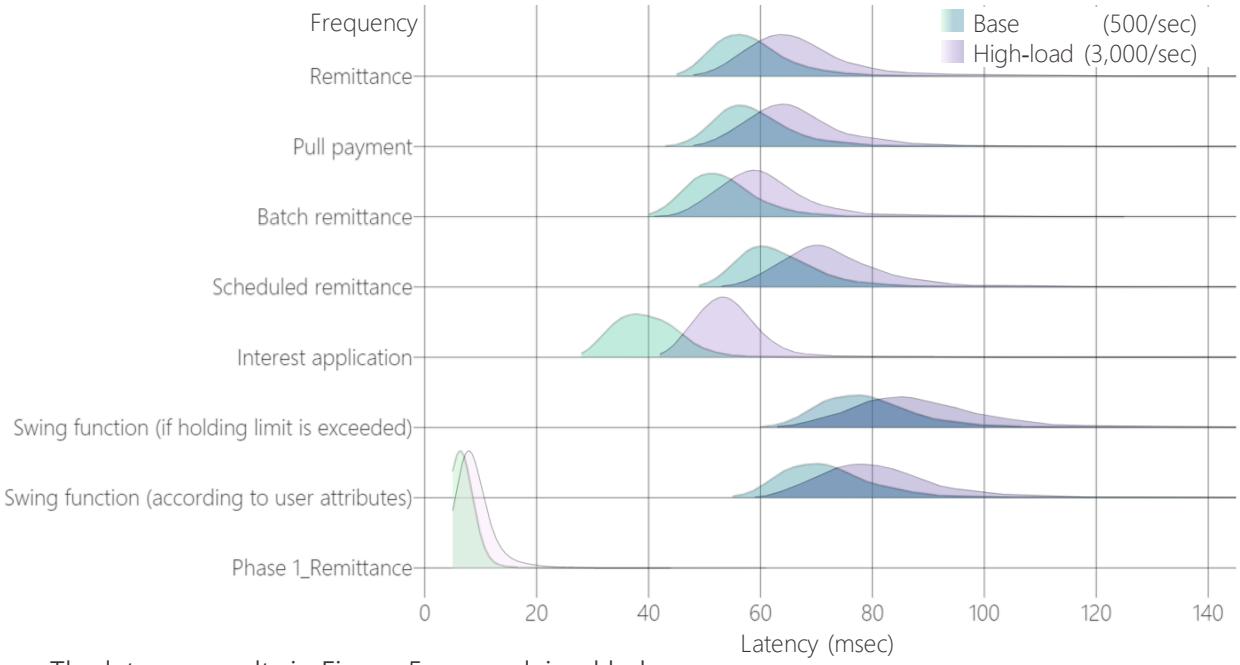
Performance evaluation test

To evaluate the impact of implementing additional functions on system performance, the basic settings for the test were kept the same as in Phase 1. Specifically, the number of users was set to 100,000, the number of intermediaries was set to 5, and the load for the basic function of transaction instructions was set to 500 transactions per second for the base scenario and 3,000 transactions per second for the high-load scenario.

The test results showed that the throughput (number of transactions processed per second) achieved the same rate of requests submitted per second and that no performance bottlenecks, including those caused by resources, were encountered. Compared with Phase 1, latency (time to process one transaction) increased somewhat, with its distribution having larger deviation due to the increase in the complexity of processing and the number of systems involved. Overall, however, no significant performance degradation was observed in latency, generally achieving less than 200 milliseconds (msec)/0.2 seconds (sec) or lower, even at high-loads (see Figure 5 for the latency results and Appendix 2 for details of the test methods and results).

[8] In the case of scheduled remittances and batch remittances, where multiple remittances are executed, the batch processing method involving the processing of multiple remittances at once was not adopted; rather, each remittance was split and processed one at a time so as to minimize the impact on other transaction processes.

Figure 5: Latency results by function[9]



The latency results in Figure 5 are explained below.

Remittance

The latency is about 60 msec, which is slightly higher than that of the Phase 1 remittance because of the increased number of systems and processing content due to additional functions; however, it does not result in significantly lower performance.

Pull payment

Although the origin of the instruction is reversed (from payee), the latency is almost the same as that of a standard remittance because the transaction processing is the same except for being kicked by payee.

Batch remittance

To obtain multiple sets of measured values within the test time, we set 600 remittance instructions as a batch. We then implemented a method in which the upper limits on holdings and transaction amount/number were obtained for a batch of 600 remittance instructions at once; next, the remittance instructions were split and processed one by one. The latency per remittance after the split was lower by several msec than that of a standard remittance because the upper-limit values were obtained at once.

[9] Latency is measured as the time taken from the start to the end of application processing for one transaction request. To visualize the approximate trend of latency with each function, the vertical axis of the figure shows the frequency of latency, the density estimated by kernel function (using the bandwidth with reference to Silverman's test results).

Scheduled remittance

Instructions are sent from the scheduled transaction management system to the orchestration system one by one at a specified time, after which the same process as for a standard remittance is executed. The latency is higher by a few msec than for a standard remittance only for the portion starting from the scheduled transaction management system.

Interest application

Although implemented using the scheduled remittance mechanism, the latency is slightly higher than that of a scheduled remittance because it is not subject to transaction amount/number limit.

Swing

The latency is slightly higher than that of a standard remittance because of the additional converting process to bank deposits and so on.[10]

Performance prediction of Design 2

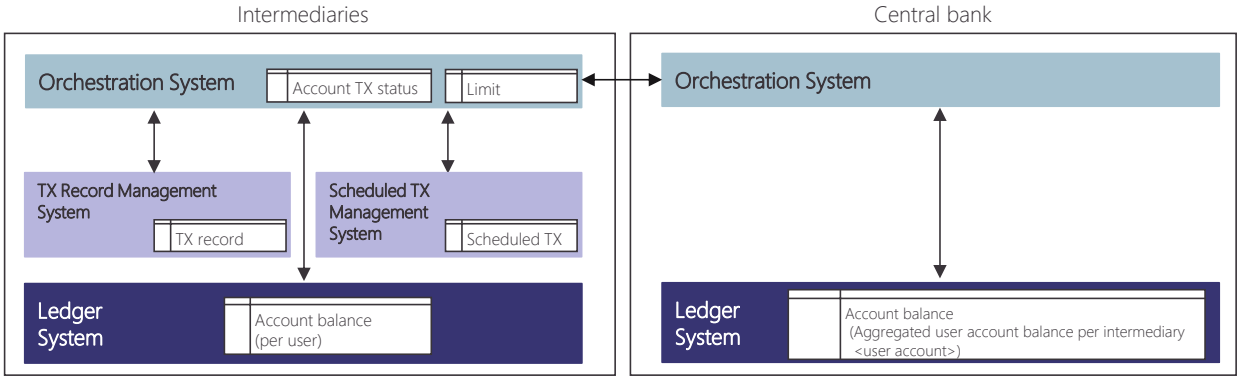
Given that the architecture in Design 2 is so designed that user accounts are managed in each intermediary's ledger, user limits and transaction record, which are necessary for the various limit checking processes, could also be managed by the intermediaries (see Figure 6).[11] In this case, the various limit checking processes performed in the central bank system in Design 1 are assumed to be performed in the system of the respective intermediaries in Design 2.[12]

[10] The latency of swing function according to user attributes was slightly lower than the one of swing function if holding limit is exceeded due to the absence of a calculation process for the amount exceeding the upper limit.

[11] In this case, the central bank's orchestration system will retain only the point-of-contact function as a facade between the systems. In the central bank's ledger system, as in Phase 1, the update process occurs only when the user accounts (the aggregated CBDC balances of users per intermediary) change mainly because of transactions between users across intermediaries (in the case of remittances within the same intermediary, the update process does not happen because the user accounts of the intermediary on the ledger does not change).

[12] In the case of remittances between different intermediaries, each of the payer's/payee's intermediaries performs its own transaction amount/number limit check processing, and only the payee's intermediary does the holding limit check processing.

Figure 6: Example of a system architecture for Design 2



Note: TX=transaction

The latency of Design 2, which relates to transfers between different intermediaries, was calculated under some assumptions[13] based on the system architecture described above using the results of the performance evaluation test; it was higher by several tens of milliseconds than Design 1, but this did not represent significantly lower performance. The increase by several tens of milliseconds is attributable to the following. In the case of Design 1, the ledger update process is completed only through decreasing and increasing amounts in the central bank ledger, while in Design 2, updates to the respective ledgers of the payer's and payee's intermediaries are also added.

In the central bank ledger of Design 2, the aggregated CBDC balances of users per intermediary are recorded in the user accounts. As pointed out in Phase 1, if the load volume of transfers across intermediaries increases, the update process may be concentrated, resulting in performance degradation due to the effects of record locks. Such performance degradation could be avoided or mitigated by splitting the records of the user accounts or by ensuring the update processing of account transfers on the central bank ledger and that of intermediaries are performed at different times.[14]

[13] The calculation methodology and assumptions are as follows.

The flow of remittance is broken down into individual processing units, such as checking of the transaction amount/number limit, debiting CBDC from the payer's account, and depositing CBDC into the payee's account, and the probability distribution of processing time for each individual processing unit is estimated by referring to the probability distribution of processing time for similar processes obtained in the performance evaluation test of Design 1. Next, the distribution of the sum of several stochastic variables is calculated for the parts in which each process is performed in sequence, and the distribution of the maximum value of those stochastic variables is calculated for the parts in which several processes are performed in parallel. Finally, by combining these processing time distributions, the probability distribution of latency for the entire flow is calculated. For the calculations, assumptions and preconditions were formulated, such as that the processing times of each individual process are independent of each other and that the intersystem communication time is constant.

[14] In performing the above, it is also important to examine ways to maintain the atomicity of transactions, etc., which was achieved by synchronizing central bank ledgers with the ledgers of intermediary institutions.

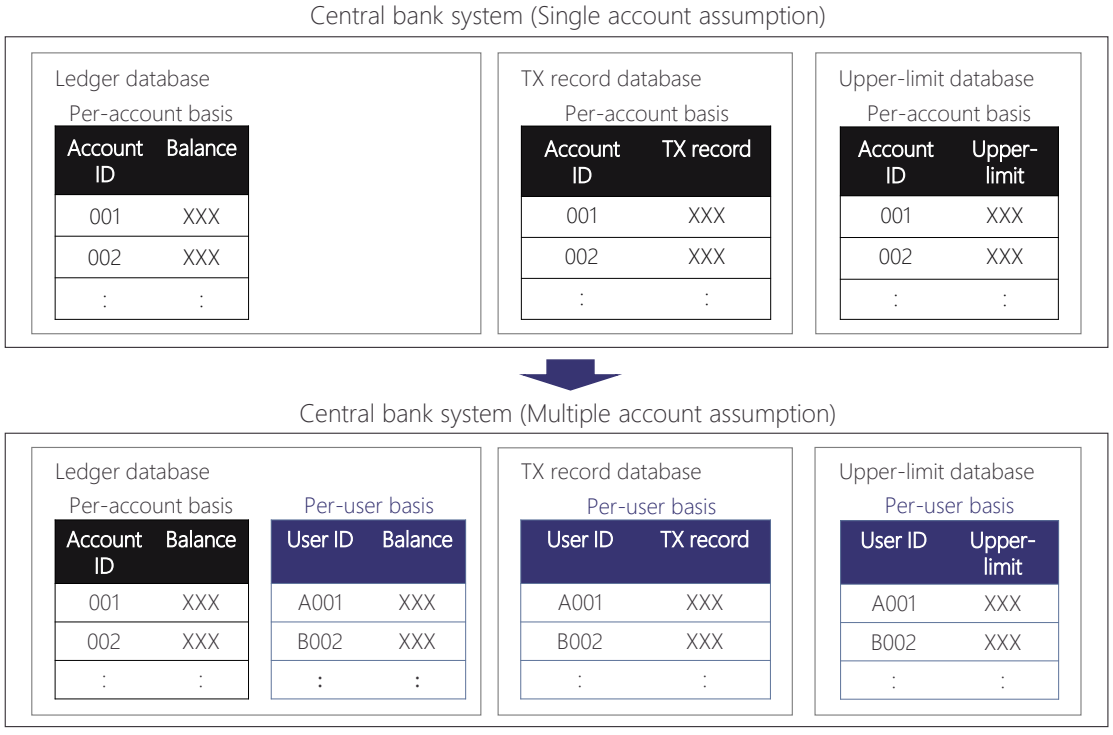
3.1.2 Coordination between intermediaries and connection with external systems

CBDC accounts provided by multiple intermediaries

In the PoCs, so far, we have proceeded on the assumption that a user has only one CBDC account with one intermediary (single account), but from the viewpoint of user convenience, a mechanism that allows a user to have one CBDC account each with multiple intermediaries (multiple accounts) would also be possible. If we allow the provision of CBDC accounts by multiple intermediaries to a single user, various limits process such as those discussed in 3.1.1 might need to be implemented on a per-user basis (based on the aggregation of multiple accounts for a given user). We explored means of coordination between systems to enable this possibility.

In the case of Design 1, because user account balances and transaction records are managed in the central bank's system, it is feasible to implement various limits on a per-user basis by changing the granularity of information managed from an account-by-account basis to a per-user basis (see Figure 7). In this case, the processing is essentially the same as when a single account is assumed.[15]

Figure 7: Change in the information management granularity for Design 1



Note: TX=transaction

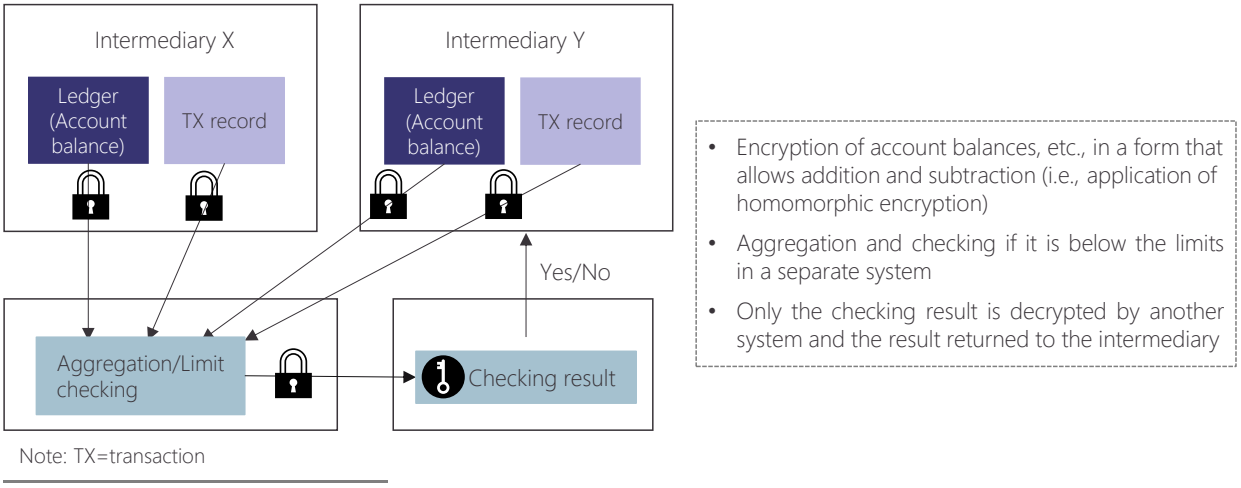
In Design 2, because the balance and the transaction record of each user's account are managed separately in each intermediary's ledger, if various limits are to be imposed on a per-user basis on

[15] As shown in Figure 7, only the ledger database is expected to experience the addition of a very small amount of processing time because it is necessary to maintain not only the per-user but also the per-account data tables and update both.

the assumption that a user has multiple accounts, the intermediary that performs the upper-limit checking process may need to collect information such as the balance information of the relevant user that is held by other intermediaries.

From a privacy perspective, it might be undesirable for user balance information, etc., held by an intermediary to be shared with other intermediaries; a coordination method that takes this into consideration is having a separate system for collecting information necessary for the limit checking process.[16] In this case, for example, homomorphic encryption, which enables data processing in an encrypted state, could be used to enable information collection and limit check processing in a separate system while maintaining the confidentiality of the information[17] (see Figure 8). The simulation results show that if such a system architecture were adopted, the processing time for encryption and decryption of remittance-related data would be about 50 milliseconds.[18]

Figure 8: Example of homomorphic encryption use in Design 2



Note: TX=transaction

[16] Another possible system architecture is having no separate system for information collection but having each intermediary complete the limit checking process based on the microservices concept. See Appendix 1 for details.

[17] Further, if it is required that user data (e.g., user ID, account ID, etc.) other than monetary information such as account balances and transaction records could be retrieved in encrypted form, searchable symmetric encryption could be used. There is also a possibility of using information concealing technologies such as multi-party computation (MPC), which divides data and performs computation on multiple servers, and trusted execution environment (TEE), which performs processing in a segregated hardware area. For more information on each of these technologies, see the following reference: Payment and Settlement Systems Department, BOJ, "Privacy Enhancing Technologies: Payments and Financial Services in a Digital Society" (January 2023, Payment and Settlement Systems Report Annex Series).

[18] To calculate the processing time when homomorphic encryption is applied, we used the Paillier encryption, which relatively has many examples of practical use, with several variations in key length. Further, we conducted a simulation in which encryption, addition, subtraction, and decryption were performed 10,000 times each in a different environment with the same specifications as those of the experimental environment. The simulation results show that the total time for encryption, addition, subtraction, and decryption was 46 milliseconds even with a key length of 2,048 bits, which is currently considered highly secure. In addition to the Paillier encryption, there are other homomorphic encryptions such as ElGamal and RSA encryptions, which offer different computational processing capabilities and speeds. Although Paillier encryption can perform only addition and subtraction, they are considered to work relatively fast, so there are many examples of social implementations that use Paillier encryption.

The above results suggest the possibility of assuming multiple accounts in Design 2 and implementing various limits on a per-user basis without significantly degrading remittance latency while still providing a degree of consideration to privacy. However, the increase in processing at the time of remittance increases the number of possible points of failure and the probability of potential data inconsistencies. Therefore, as a measure to control the amount of CBDC circulating on a macro scale, a simpler method instead of the method described above is to set an upper limit on the number of accounts held by a single user and on the amount held per account and the amount and number of transactions so that there is no need to aggregate balance information, etc. for each account.

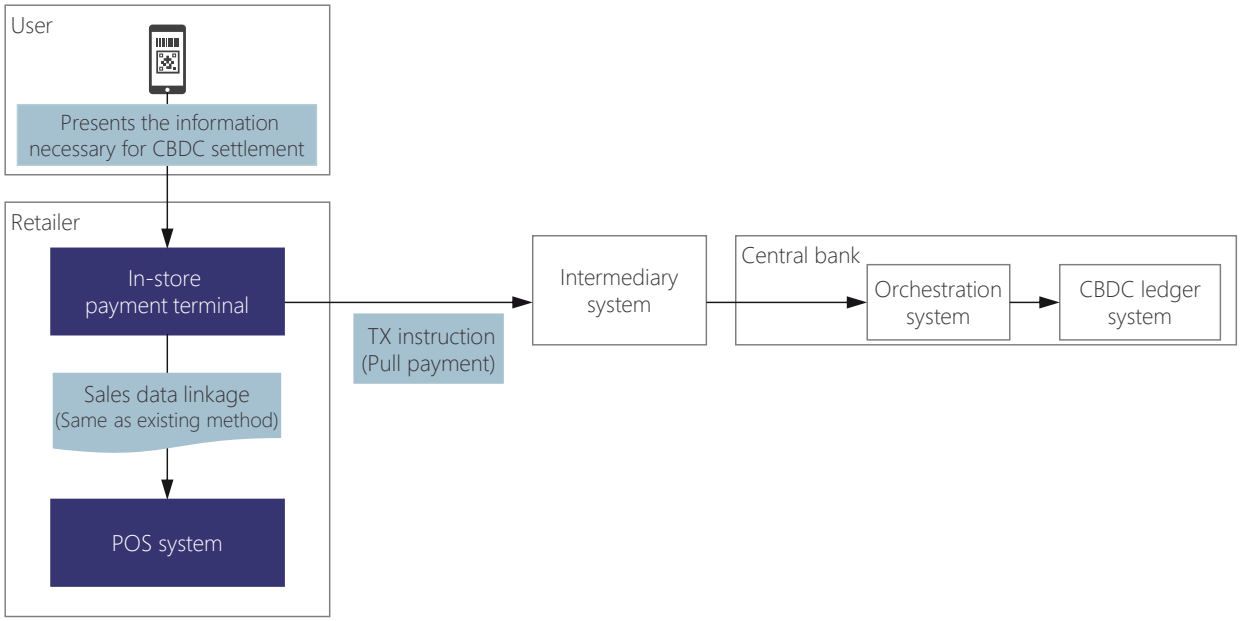
Connection with external systems

There are a variety of possible external systems that may need to connect with CBDC systems. For example, if we consider connection with a point of sales (POS) system for in-store payments, we could use existing connection methods between the store's payment terminals and the POS system.

Many retailers, such as department stores, supermarkets, and convenience stores, currently have POS systems that manage data on daily sales and products sold, and each time a payment is made in cash or with electronic money, the store's payment terminal processes the received payment and automatically links the generated sales data (e.g., products, quantity, etc.) to the POS system. Such automatic connection could be used when CBDC payments are made at retail stores in the future. For this application, it is assumed that stores' existing payment terminals will be modified to support CBDC payments while also ensuring the confidentiality of payment information.

As an example of connection with a POS system, when a user pays with CBDC at a store, the store's payment terminal (which supports CBDC) instructs the pull payment by CBDC and links sales data to the POS system. This case is illustrated in Figure 9.

Figure 9: Example of connection with POS system for in-store settlement

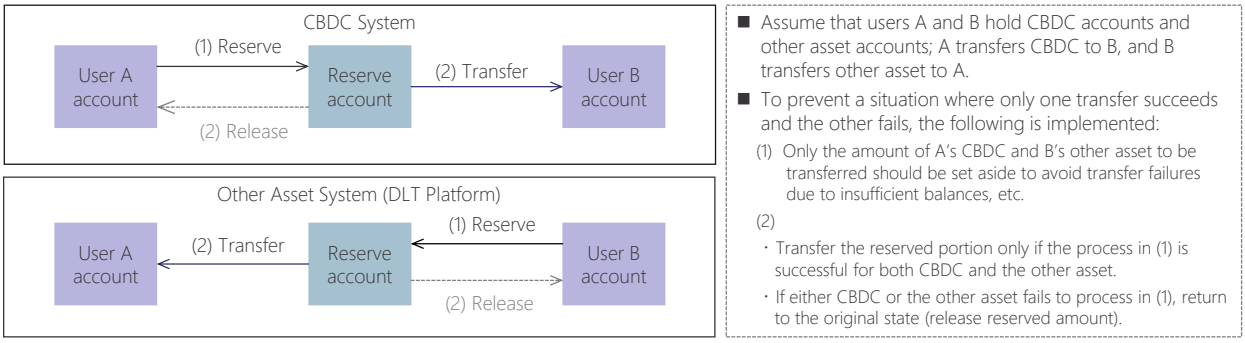


Note: TX=transaction

The external systems that may connect with CBDC systems could be based on not only existing technologies but also new ones, such as distributed ledger technology (DLT). Many CBDC experiments by other central banks have dealt with the connection between CBDC systems and DLT platforms with an eye toward "asset tokenization," in which assets are tokenized and distributed via a DLT-based platform.[19] Therefore, this PoC explored the exchange of CBDC and assets on DLT platforms. Referring to existing examples of delivery versus payment (DvP; simultaneous settlement of securities and funds) and payment versus payment (PvP; simultaneous multicurrency payment) settlements, we assumed a linkage processing in which the assets to be exchanged are reserved once in each system and the final exchange is performed after they are successfully reserved in both systems, thereby providing escrow (Figure 10).

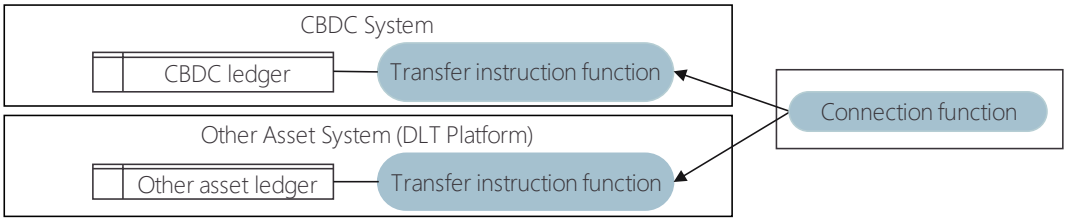
[19] For examples of foreign central banks' CBDC experiments that use DLT, see the following reference: Jiro Sugie and Junichiro Hatogai, "Efforts to Improve Settlement Using Distributed Ledger Technology - Focusing on Wholesale CBDC Experiments in Different Countries" (November 2022, Bank of Japan Review Series, 2022-J-16, available in Japanese).

Figure 10: Example of a linkage processing



A possible method for implementing a linkage processing is for the system that manages the assets to be exchanged to have an application that issues asset transfer instructions and to connect them indirectly via a connection system that is independent of both systems (the connection system would have an application that triggers transfer instructions at the appropriate time; see Figure 11). Regarding the application of the respective systems that perform the connection function (the transfer instruction function and the connection function portion in Figure 11), the system performing the connection function can also perform the transfer instruction function; further, other connection methods are conceivable because the architecture and characteristics of the DLT platform vary.

Figure 11: Example of a system architecture



Regarding such interoperability with external systems, one option would be to make effective use of existing system infrastructure and technological connection methods. The external system could be assumed to belong to either the competitive or non-competitive area, depending on their function. Therefore, it is also important to explore this from a variety of perspectives, including those on institutional arrangement.

BOX: Points to consider in offline payments

In the PoCs, so far, we have assumed a situation in which wallet applications, intermediary and central bank systems, etc., are connected online via networks at all times. As a future use case of CBDC, it is possible to envisage a settlement in which CBDC is transferred between user terminals (P2P connection) in an offline environment, even temporarily, without the need for authentication or connection to a ledger server, etc. For such offline payments, it is important to have a "fraud/inconsistency detection" function. On considering the following three possible cases of fraud/inconsistency, we see that each one has its own set of issues that require further examination (the data model for offline CBDC is assumed to be either a fixed-value or a flexible-value token in this study).

1. Cases in which offline CBDC is generated fraudulently

   When an offline CBDC is generated by a stakeholder who does not have the authority to generate it

   - One solution is to add a digital signature from the central bank to the offline CBDC stored on the terminal and verify the signature at the receiving user's terminal at the time of settlement.
   - In this regard, especially in the case of flexible-value token, where the tokens are merged or split as needed for each transaction, it is necessary to consider separately how to check the legitimacy of the tokens generated on the user's terminal, etc.

2. Cases in which offline CBDC is replicated across different terminals

   When a legitimate offline CBDC on one user terminal is replicated on another terminal

   - It is necessary to explore designs that could be considered to detect when a transfer is instructed for a replicated CBDC.

3. Cases in which used offline CBDC remains in the terminal and is used twice

   When a legitimate offline CBDC on one user terminal remains stored on the same terminal in an unused state after an offline remittance and is used again during another offline remittance

   - In the case of no connection to the ledger server or other online environments, it is technically difficult to detect double use on each offline payment at the receiving user's terminal. The same issue has been noted in experiments conducted in other countries.
   - There is scope to explore an other method in this respect, for example, by connecting each terminal to an online environment on a regular basis to collect and collate the necessary information and check after the fact whether there has been any double use.

It is necessary to eliminate user fraud, etc., by ensuring that terminals and applications on terminals are robustly constructed so that the kind of fraud/inconsistency described here can be avoided. However, even assuming that terminals and other devices are robustly constructed, the possibility of their having latent defects or unknown vulnerabilities cannot be completely eliminated. Therefore, even if fraud/inconsistency does occur, it is necessary to explore mechanisms to detect when it is about to be used for offline payment.

It is important to consolidate assumed situations and assumptions and continue technical evaluation for offline payments. It is necessary to focus on not only the technical evaluation process but also the privacy perspective when proceeding with further consideration of potential measures.

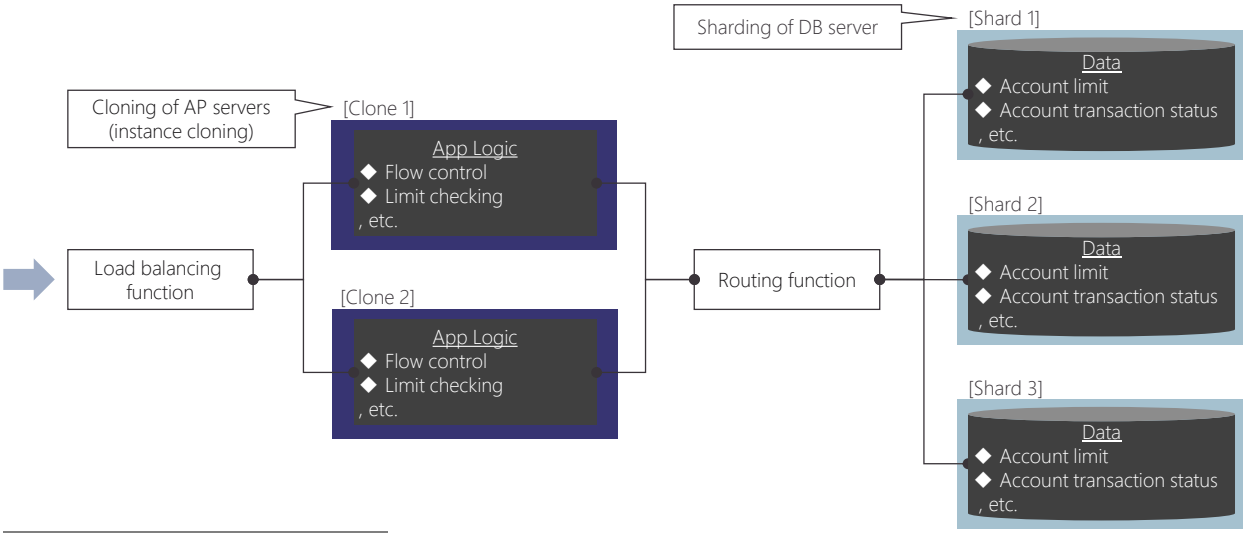
3.1.3 Issues in the possible event of social implementation

Scalability

In the possible event of social implementation, as envisioned in Phase 1, the additional functions must be scalable to handle large-scale, high-frequency processing of tens of thousands to hundreds of thousands of transactions per second and enable the management of vast amounts of account information.

For performance enhancement of the experimental environment, cloning (instance replication) is a possible method for load balancing and parallel processing by deploying multiple identical servers for the application (AP) server, which executes the processing of transaction instructions. As for the database (DB) server, which records and holds the results of the processed instructions, the first step required is to scale it up; however, because scaling up is limited by the hardware specifications of servers, horizontal partitioning (sharding)[20] is the other way to reduce the load (Figure 12). In other words, AP servers, which implement application logic for the execution of processing, can be configured to process instructions independently for each server, so a certain level of performance enhancement can be achieved by simply increasing the number of servers. A DB server, managing a huge number of records, can be horizontally partitioned to control the amount of data and access for each shard, thereby ensuring sufficient processing performance. In addition to the need to achieve consistent processing among multiple servers when performing horizontal partitioning, there are system operation issues such as the need for periodic tuning to ensure that the amount of data and accesses held within the server are consistent.

Figure 12: Performance enhancement approaches (example of an orchestration system)



[20] A method in which one set of table data stored on a single DB server is distributed and maintained on multiple servers on a per-record basis.

Regarding maintainability when additional functions are added or modified, it is important to explore modularization (componentization of processes that lead to improved development and modification efficiency) of processing in relevant areas because the impact of the modification will be concentrated on the orchestration system in this particular configuration.

Reliability

To achieve high availability, it is important to create redundant systems and prepare disaster recovery (DR) sites for disaster countermeasures to increase fault tolerance. In Design 2, even if the central bank system were to fail, the impact on processing that is completed only by the intermediary institution system would be limited, although the number of points of failure could be large. Further, if the provision of accounts by multiple intermediaries and various limits per user are assumed, any failure in a system that centrally aggregates balance information, etc., could affect not only the payer's and payee's intermediaries but also other intermediaries holding accounts for the user. Therefore, there is considerable potential for exploring ways to address this issue, including an institutional arrangement that would eliminate the need for aggregation of balance information, etc. by setting an upper limit on the number of accounts and imposing various limits on each account, as described above.

The security measures in the possible event of social implementation are the same as those in Phase 1. In other words, measures need to be taken for each component of the CBDC system, such as individual servers and inter-system networks. It is important to consider finding an appropriate security balance, keeping in mind the following trade-off relationship: the higher the required level of security, the greater the cost of implementing and operating the measures and the lower the processing performance.

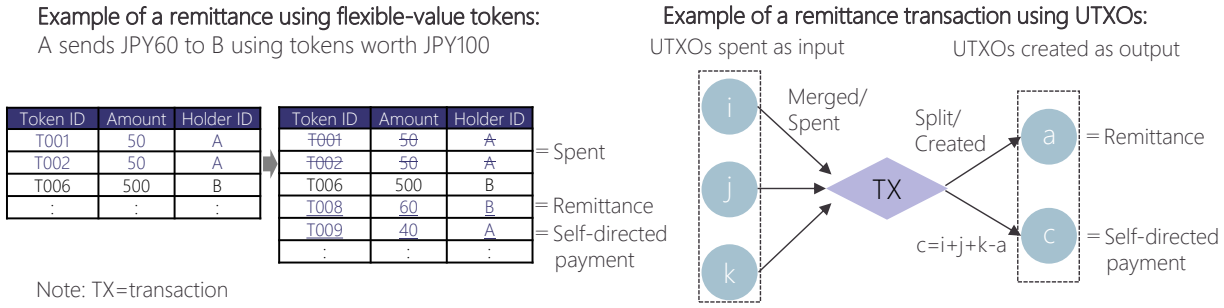
3.2 Potential use of new technologies

3.2.1 Flexible-value token model

Although a token-based data model can be interpreted in various ways depending on the context, in the PoCs, the model is one in which a unique identifier (ID) is assigned to monetary data with a face value, and the CBDC holding status can be recognized by linking this ID with a user ID.

As indicated in Figure 3, for the ledger design alternatives to be evaluated, the fixed-value token was considered for Phase 1, in which the face value is fixed like cash, and the user to whom the existing tokens are linked is changed after they are converted to ones with lower value as necessary at the time of remittance. In Phase 2, a flexible-value token model was considered, in which existing tokens are merged or split as necessary at the time of remittance, thereby inducing a change in their face value and linking the user to the newly created tokens. In the flexible-value token model, existing tokens are merged and spent as inputs and two separate output tokens are newly created for remittance and change (self-directed payment); the new token for change (self-directed payment) are used as inputs for the next remittance. When considering how the tokens are spent during remittance, the flexible-value token model is a data model similar to the unspent transaction output (UTXO) model[21] (Figure 13).

Figure 13: Example of remittances using flexible-value tokens and UTXOs[22]

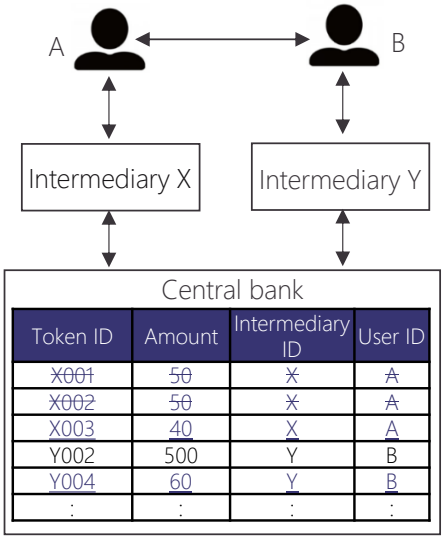


Regarding the method of implementation of a flexible-value token model, first, consider the case of Design 3, in which the central bank alone manages the ledger. The payer's intermediary could select the tokens to be spent for the transfer, calculate the change, generate new tokens, and execute a transfer request to the central bank, which would then update the ledger (Figure 14).

[21] In the UTXO model, information on past output is often recorded in a form that links spending to input, but in this flexible-value token model, only information on unspent output tokens is recorded.

[22] The illustration of the UTXO remittance transaction was prepared with reference to ECB "Documents for the Digital Euro Prototyping Exercise" (December 2022), Annex 1.

Figure 14: Example of Design 3 (centralized management by the central bank)



Example: A sends JPY60 to B using tokens worth JPY100

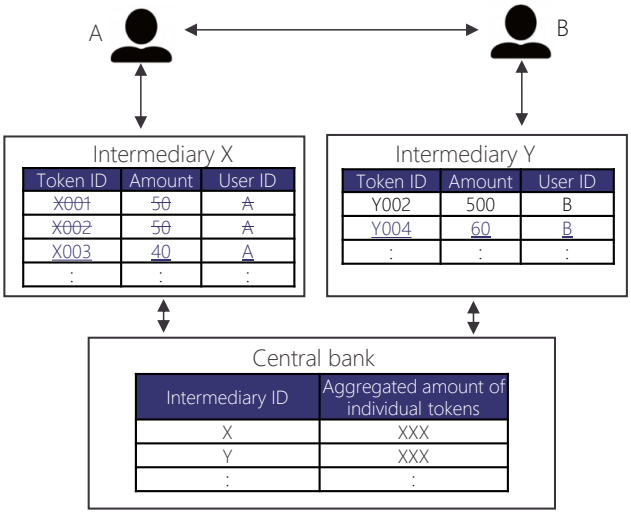
- The payer's intermediary, X, does the following:
  - ✓ Select tokens to be used for remittance (X001, X002)
  - ✓ Calculate self-directed payment (40)
  - ✓ Create new tokens (X003,Y004)

  This information is sent to the central bank.
- The central bank updates the ledger.

In the case of Design 4, in which the central bank and the intermediaries share the management of the ledger, it is assumed that the individual token information held by the user is managed by each intermediary. In this case, tokens transferred across intermediaries are assumed to be numbered and created by the payer's intermediary, and the payee's intermediary takes over the management of the tokens after the remittance (tokens remaining with the payer's intermediary will continue to be managed by the payer's intermediary).

Under the assumption about intermediaries described above, the central bank could have a role of managing the aggregate token amount on a per-intermediary basis as shown in Figure 15. Although the updating process would be concentrated in the central bank ledger, the performance advantages of the token-based data model, capable of processing multiple requests in parallel, could be maintained in the ledgers of the intermediaries if the same solutions as in Design 2 are applied, for example, the split records in the central bank ledger, as described in 3.1.1.

Figure 15: Example of Design 4 (shared management between central bank and intermediaries) ①



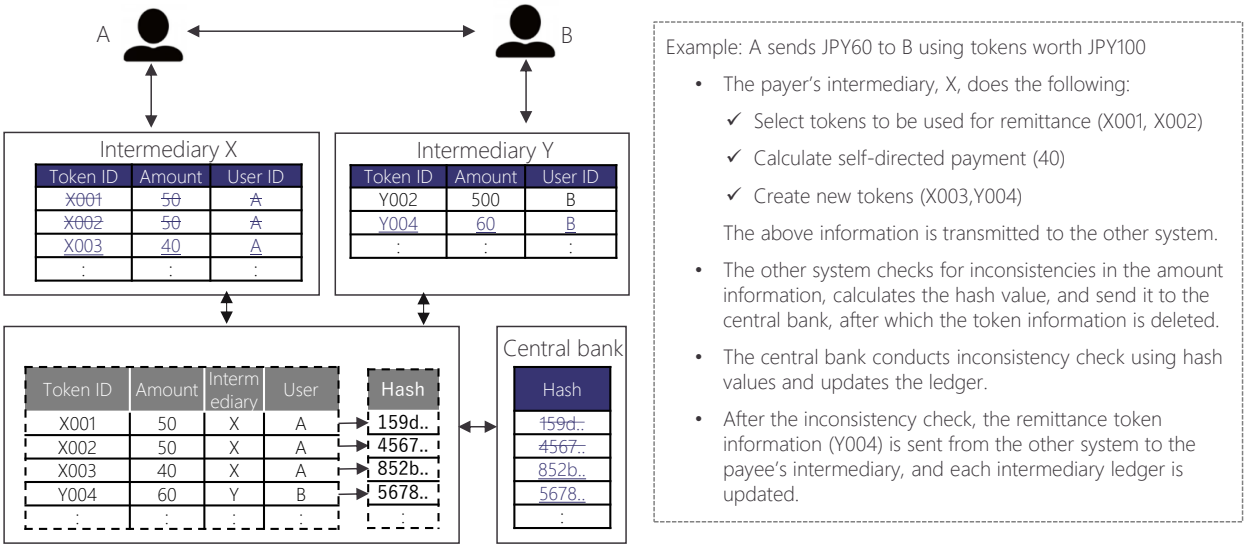
Example: A sends JPY60 to B using tokens worth JPY100

- The payer's intermediary, X, does the following:
  - ✓ Select tokens to be used for remittance (X001, X002)
  - ✓ Calculate self-directed payment (40)
  - ✓ Create new tokens (X003,Y004)

  The above information is shared with the payee's intermediary via the central bank.
- The payer's and payee's intermediaries update individual token ledger information.
- The central bank updates only the aggregated amount of individual tokens by intermediary.

Another possible architecture is the central bank partially fulfilling the role of maintaining the consistency of the tokens circulated among intermediaries as a whole but not holding details of individual token information. One such example is an architecture in which the token amount information transmitted from the payer's intermediary is checked for inconsistencies in a separate system, a hash value is calculated (after calculation, the token information is deleted), and only the hash value[23] is transmitted to the central bank, which then detects double spent of tokens (Figure 16).

Figure 16: Example of Design 4 (shared management between the central bank and intermediaries) ②



In both Designs 3 and 4, there are multiple possible approaches, but in either case, the number of tokens to be updated at the time of remittance under a flexible-value token model will be correspondingly smaller because the exchange process required in the fixed-value token model is no longer necessary. Therefore, the flexible-value token model is likely to take advantage of the performance feature of being able to process multiple requests in parallel. However, the degree of performance may vary depending on the merging and splitting algorithms and the resources required may increase compared to the account-based data model; further, the difficulty of implementing additional functions such as holding limits while maintaining performance may also increase.

[23] A hash value is calculated as the output of a function (cryptographic hash function) that obtains a fixed-length value from arbitrary input values with no regularity. The hash value has several advantages; for example, it can detect falsification of lots of different input information collectively (the output hash value changes if the information used in the input differs even slightly); it is not necessary to hold the user's token information directly (only the hash value of the irreversibly converted input information needs to be managed).

3.2.2 NoSQL Databases

We used a traditional RDB as the system database in the experimental environment from Phase 1 onward. However, because a CBDC system requires extremely high processing performance, in Phase 2, NoSQL -- databases other than those based on RDB, some of which are said to have high processing speed and performance scalability -- were explored for possible use.

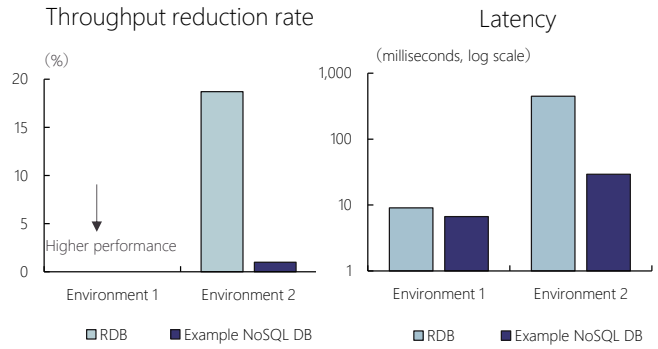
In the experimental work, performance evaluation tests were conducted using a key-value[24] type NoSQL database as an experimental setting for an account-based ledger system built in the public cloud. For the purposes of performance evaluation, implementation was designed to ensure the necessary consistency,[25] including the use of transaction functions considered necessary for settlement operations (in this case, a function that simultaneously commits two records in a single transaction: one for a decrease in the payer's account and one for an increase in the payee's account). High-load scenarios were used for two environments with different degrees of processing concentration: Environment 1, in which updates occur almost randomly to 100,000 user accounts as in the central bank ledger of Design 1, and Environment 2, in which updates are concentrated in the aggregated user accounts held by five intermediaries as in the central bank ledger of Design 2. The test results showed that all NoSQL databases maintained the necessary consistency even with the concentration of updating process and that in-memory databases[26] in NoSQL databases could improve processing speed (Figure 17).

[24] There are various types of NoSQL databases and data models that are used within them. The performance evaluation tests were conducted in the experimental environment of an account-based ledger system; a NoSQL database of the key-value type, which is a data model of a simple combination of a unique key that can identify data and a value, specifically designed for processing rows of data, was selected for the test.

[25] In addition to the transaction function, we implemented a function in the application that allows retry processing when, for example, multiple data update processes conflict with one another, resulting in an error. In other words, the NoSQL database used in the experimental work did not utilize the record lock function that is common in RDBs, but controlled consistency by monitoring the data update status; the default specification is that when data update processes conflict, the subsequent process returns with an error and is not re-executed. Therefore, we implemented a mechanism in the application in which subsequent processes are re-executed as retries when data update processes conflict.

[26] A database that manages data in random access memory (main system memory). It enables faster data reading/writing than conventional databases that manage data on disks.

Figure 17: Test results



Environment 1: Central Bank ledger in Design 1
(Without concentration of updating process)

Environment 2: Central Bank ledger in Design 2
(With concentration of updating process)

Note: All figures are at high load (3,000 transaction requests/second).
- Throughput reduction rate is expressed as the rate of decline in the number of transaction requests processed relative to the number of transaction requests injected over a second.
- Latency is AP processing time + DB processing time.
- All values are averages for the evaluated period.

The architecture evaluation based on the test results suggests the possibility that some databases could maintain high-performance scalability even under the load expected in the possible event of social implementation, implying that they could be used in ledger systems.[27] The use of transaction functions may hinder performance scalability through restrictions on sharding, and some databases may require further refinement for use in ledgers. However, the evaluation also suggest that even such databases have the potential for use as databases for CBDC systems that handle data where transaction functions are not required (e.g., account transaction status management for the implementation of additional functions, replica databases for balance inquiries).

There are many other types of NoSQL databases in addition to those evaluated in this PoC, and even RDBs are now beginning to emerge as high-performance databases. Depending on the expected volume of transactions, there may be more advantages to build a stable system using traditional databases. The possibility of using new databases needs to be explored from perspectives other than performance (e.g., ensuring data integrity in case of failure in an in-memory database[28]). Going forward, the ever-evolving database technologies should be explored from a broad perspective continuously, considering the requirements of anticipated transactions and other factors.

[27] The experimental work was conducted on an account-based ledger system. It may be more difficult to accurately estimate the total number of database records required for social implementation of a token-based ledger system than an account-based system, because of the exchange, merging, and splitting processes involved. From this perspective, the ease with which the performance of NoSQL databases is enhanced suggests that a token-based type model may be more advantageous than an account-based type.

[28] A common approach to enhancing data integrity is to record data in some form of data storage. Some measures come with potential performance tradeoffs including additional processing time.

4 Conclusion

In PoC Phase 2, we evaluated the processing performance and technical feasibility of additional functions for which it would be desirable to confirm technical issues as early as possible in the possible event of social implementation. Further, the possibility of utilizing new technologies that had not been examined in Phase 1, such as data models and databases, were also evaluated.

In evaluating additional functions, we examined these in terms of (1) preempting a sudden shift from bank deposits to CBDC as safeguard, (2) improving user convenience such as scheduled and batch remittance and pull payments, and (3) enabling coordination among intermediaries and connection with external systems. The evaluation results suggest that none of these additional functions would cause significantly lower performance and could ensure the scalability and reliability required for social implementation; they also indicate the need for further considerations and solutions to achieve the implementation.

In the evaluation of new technologies, we found that the performance advantage of token-based data model for ledgers in terms of their ability to be processed in parallel might be more easily utilized in the flexible-value approach than the fixed-value approach for the system architectures considered in this PoC; however, the resource requirements and difficulty of implementing additional functions might increase compared with the account-based data model. While NoSQL databases have potential for use as databases for ledgers and non-ledger CBDC systems from the perspective of performance improvement, it was confirmed that there were differences in the degree of potential use depending on the assumed operational and other requirements and that this topic needed to be explored from perspectives other than performance.

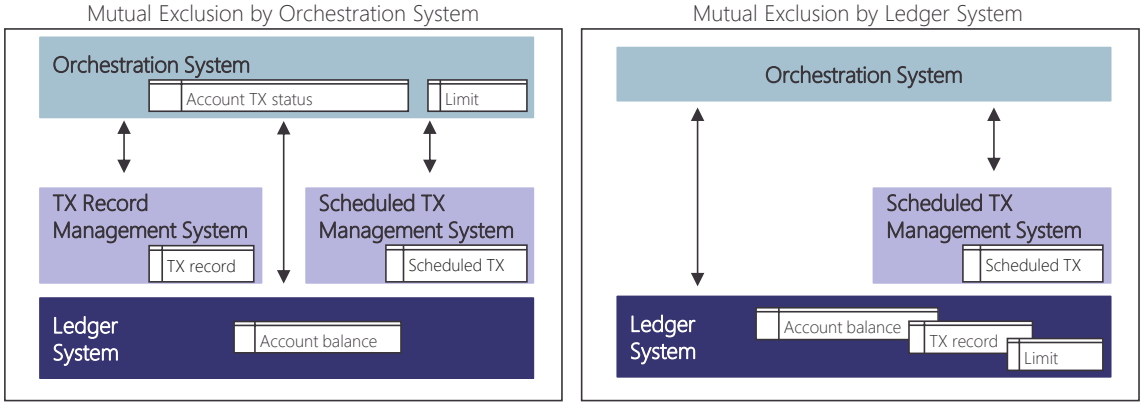
The PoCs, through which the BOJ has confirmed the technical feasibility of the basic functions of a CBDC, were completed in FY2022, as initially scheduled, with the achievement of its desired objectives. We have launched in April 2023 and are proceeding with the pilot program. In the pilot program, the end-to-end process flow will be tested, the measures and potential challenges for connection with external systems will be explored, and considerations and solutions indicated as necessary in the PoCs will also be explored. Further, a CBDC Forum will be established with a view to proceeding with institutional arrangements for CBDC in an appropriate manner, and ideas and insights set out by private businesses related to retail payments will be drawn upon to deepen the study.

Appendix 1: Alternative system architectures

In implementing the additional functions, several systems were added to the CBDC ledger in the experimental environment; one of these -- the orchestration system -- was used to ensure data consistency among the systems. The orchestration system has a DB that records and manages the status of whether or not the processing of each account is being updated; if it is, in principle, the processing of other operations is put on standby,[29] thus allowing the application to control the order of processing.

A possible system architecture that differs from the system used in the experimental environment is having the record and other information on the ledger system and consolidating the implementation of additional functions. In this case, the mutual exclusion by the ledger DB alone (the so-called record lock function[30]) can be used to ensure consistency in processing. Specifically, the ledger system performs upper-limit checking processes by directly placing information that is highly relevant to the various limits checking processes, such as transaction record and upper-limit values, in a separate table in the ledger DB; the consistency of processing is ensured by the DB's mutual exclusion (see Appendix Figure 1.1).

Appendix Figure 1.1: Mutually exclusive control method (example of Design 1)



Note: TX=transaction

If this method were used in the experimental environment, the account transaction status management within the orchestration system and transaction record management systems would become unnecessary, so that the communication time across systems would be reduced, and the processing time could be shortened by several tens of milliseconds. Additionally, the simpler system

[29] Some combinations of operations that do not interfere with the limit-check processing are processed in parallel; see footnote 5 for details.

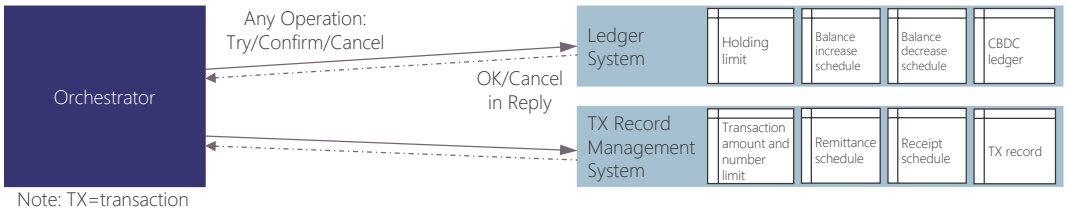
[30] For details on the function of DB record locks, see Appendix 3 of the "Central Bank Digital Currency Experiments Results and Findings from 'Proof of Concept Phase 1'."

configuration may facilitate easier recovery in the event of a failure. However, because the ledger system handles data such as transaction records and upper limits, the load on the ledger system may increase depending on the expected volume of such data and the corresponding load of transaction instructions.

Other possible alternative system architectures include a design based on the idea of microservices, in which the system is divided into units of closely related operations. Specifically, the CBDC ledger system could be configured to perform operations related to balances (updating balances and checking holding limits), and the transaction record management system could be configured to perform operations related to transaction record (updating transaction record and checking transaction amount/number limits).

A specific architecture could be as follows. The orchestrator controls the processing between systems, instructs each system reserve (Try) the balance increase/decrease and remittance/receipt schedule, and determines whether the update is possible. Each system performs a reservation and returns an OK to the orchestrator if it is able to update the schedule and returns a Cancel error message if it is unable to update the schedule. If all systems are OK, the orchestrator instructs each system to confirm the contents of the reservation, and if any of the systems are not OK, the orchestrator instructs each system to cancel the contents of the reservation. Such a processing method (Try/Confirm/Cancel or "TCC" model) could potentially be adopted (see Appendix Figure 1.2).

Appendix Figure 1.2: TCC model



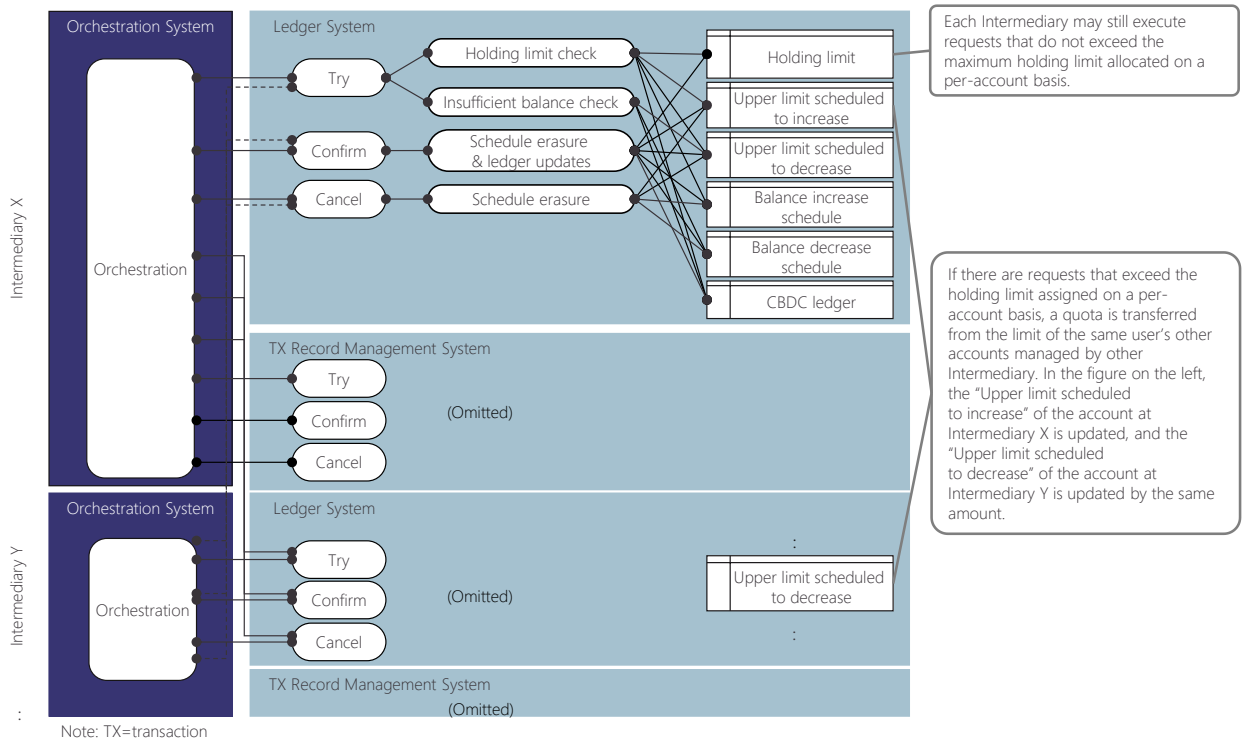
Note: TX=transaction

In the case of Design 2, where multiple accounts are assumed and various limits are imposed on a per-user basis, this method may eliminate the need for a separate system for information aggregation. For example, by assigning a limit to each intermediary's account in advance, a system in which instructions that do not exceed the limit can be executed as is, while instructions that do exceed the limit can be handled by another intermediary that manages the same user's account with a margin limit[31] (see Appendix Figure 1.3). By adopting such a mechanism to flexibly apply upper

[31] More precisely, each intermediary has a table that manages the schedule for increasing/decreasing upper limits so that the upper limits can be flexibly adjusted. For example, the upper limit increase schedule for the account at Intermediary X is updated and the lower limit schedule for the account at Intermediary Y is also updated by the same amount. If both tries are successful, even if the maximum limit of the account at Intermediary X is exceeded, the transfer will be executed by obtaining funds from the maximum allowance at Intermediary Y.

limits, it is possible to decentralize processing equivalent to the limit-check process on a per-user basis as a result, without having to consolidate user balance information, etc., in a separate system. However, there is a possibility that other intermediaries may find out that the user has exceeded the upper limit in one of the other accounts when making the margin allowance available to the user.

Appendix Figure 1.3: Example of TCC model in Design 2



Note: TX=transaction

In the case of the above configuration, compared with a case in which account transaction status management is performed within the orchestration system, in terms of performance, the processing load may be reduced because the upper-limit determination process can be closed to each system, while the processing load may increase because more data is newly updated, such as balance increase schedules and upper-limit flexibility processing. However, it is unclear which factor will have a stronger effect on performance. In terms of maintenance and operation, the same system covers related tasks; this may facilitate easier modification of the applications that complete each task.

The system architectures discussed here have extra functions in the ledger system. This is a different concept from the idea (3.1.1) that it is appropriate to have the ledger system perform as little processing as possible other than updating balances, given the possibility of a high update load required for social implementation.

Appendix 2: Details of the performance evaluation test

The experimental environment, consisted of a ledger system and other systems, has an AP server for processing transaction requests, and a DB server (RDB) for recording and maintaining the results.[32] The performance evaluation test was conducted using multiple load scenarios without changing the basic setting from Phase 1[33] to focus on the performance changes when additional functions were added.

Three test scenarios were used with some variation for the additional load associated with additional functions, as shown in Appendix Figure 2.1. In Scenario 1 (S1), only the limit checking process for various limits directly associated with transactions of basic functions (assuming that various limits are not exceeded) was added. In Scenario 2 (S2), in addition to the load volume from S1, additional functions other than interest application were added that could vary in frequency and transaction volume depending on user conditions and attributes. In Scenario 3 (S3), only the interest application, covering all users and with high transaction volume, was added to the load volume from S1; it would be expected to be processed during times when the load for the ledger is not heavy, such as at night.

Appendix Figure 2.1: Test scenarios

	Basic functions		Swing function based on user attributes	Pull payment	Batch remittance	Scheduled remittance	Interest application
	No exceeding of various limits	Swing function when holding limit is exceeded					
S1							
S2							
S3							

Base: 500/sec total; High-load: 3,000/sec total

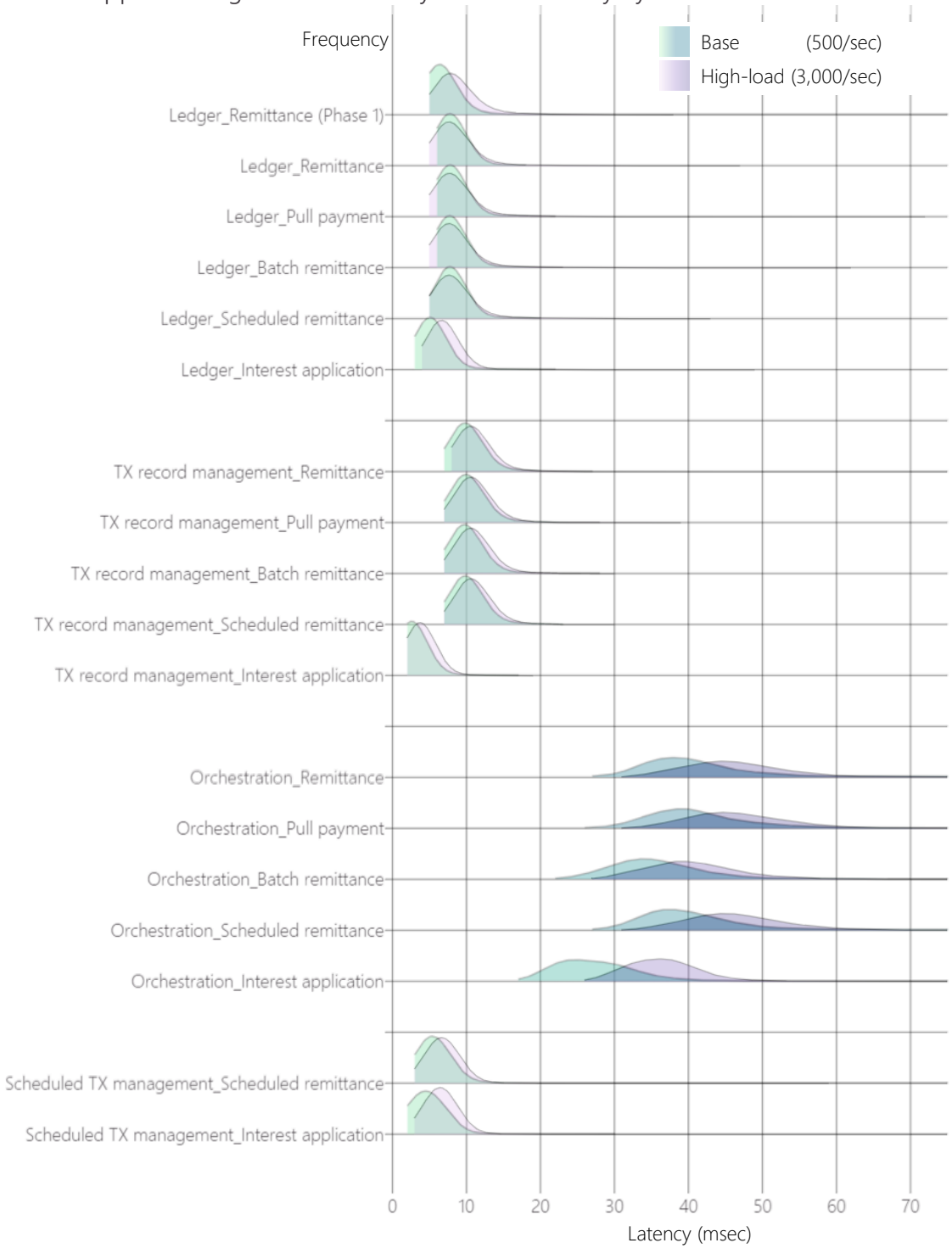
Base: 40/sec each; High-load: 240/sec each

[32] The AP and DB servers deployed in each system have 8-core vCPUs (16 cores only for the DB server of the orchestration system and transaction record management system), 64GB memory (128GB for the same systems), and SSD storage. 3 AP servers are deployed for the ledger system under the base scenario and 10 under the high-load scenario. Orchestration system: 3 units under the base scenario, 18 units under the high-load scenario; transaction record management system: 2 units under the base scenario, 10 units under the high-load scenario; scheduled transaction management system: 1 unit under the base scenario, 3 units under the high-load scenario. One DB server is deployed for all systems, regardless of the scenarios.

[33] As in Phase 1, the number of users was set at 100,000 and the number of intermediaries was set at 5, with a base scenario of 500 transaction requests per second and a high-load scenario of 3,000 transaction requests per second for the basic functions. The breakdown of transaction requests related basic functions by type is 90% for remittances (30% for ones within the same intermediary and 60% between different intermediaries); 5% each for payouts and acceptances; and 0.08% each for issuance, redemption, and balance inquiries (one transaction per minute per intermediary). The test period was also defined as the evaluation period between 5 and 10 minutes after the start of the test, when transaction instructions were continuously input for 15 minutes and stable measurements could be obtained.

The main results of the performance evaluation are described in the main text, but considering latency by system, for all function types, the level is relatively high and the shape of the distribution is flatter for orchestration systems (Appendix Figure 2.2). The latency of the orchestration system is thought to be affected by the fact that its operations include various limits checking processes, as well as inter-system communications, etc., which are comparatively prone to volatility.

Appendix Figure 2.2: Latency distribution by system and function



Note: Labels on the Y-axis show "System name_Function name." TX=transaction