Central Bank Digital Currency Experiments

Results and Findings from "Proof of Concept Phase 1"

Payment and Settlement Systems Department, Bank of Japan

May 2022

Bank of Japan

Contents

1. Proof of Concept Phase 1 objectives

The Bank of Japan (BOJ) currently has no plans to issue central bank digital currency (CBDC), but to ensure the stability and efficiency of settlement systems as a whole, the BOJ regards it as important to be prepared thoroughly to respond appropriately to any future changes in the environment, which includes undertaking technical experiments.

As the first phase of these experiments, the Payment and Settlement Systems Department of the BOJ conducted "Proof of Concept (PoC) Phase 1" from April 2021 to March 2022. The objectives for this were to build an experimental environment centered around a "CBDC ledger" as the foundation of a CBDC system, and evaluate whether the basic transactions relating to a CBDC (issuance, payout, transfer, acceptance, redemption, etc.) could be processed appropriately. To achieve these objectives, based upon the discussions with internal and external parties, the BOJ posted three design alternatives for CBDC ledgers in a public cloud.[1] While considering requirements in introducing a CBDC in the future, it then compared and evaluated each design through experimental work and architecture evaluation for its system processing performance, reliability, and ease of extension.

2. Experiment methods

2.1 Ledger design alternatives

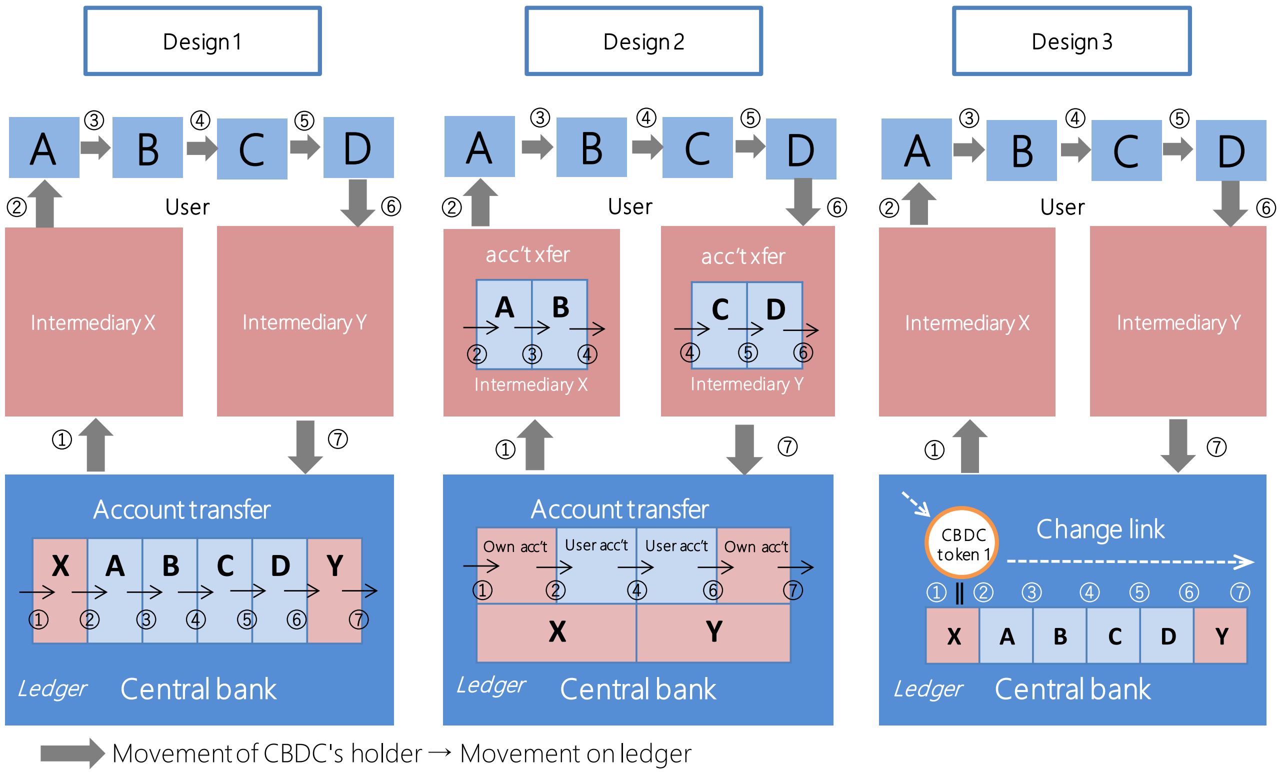
Of the three design alternatives for CBDC ledgers, Designs 1 and 2 recognized the CBDC's holding status as an intermediary's or end user's account balance, that amount to, an "account-based CBDC ledger system" (Figure 1). Of these, Design 1 is a method in which the central bank manages a ledger that records the balances of all intermediaries and end-user accounts. Design 2, on the other hand, is a method in which the central bank manages a ledger that records the account balances of intermediaries (own accounts/user accounts in aggregate), and the intermediaries each manage ledgers that record the account balances of their respective customer users. Finally, Design 3 assigns unique IDs to monetary data representing

[1] The ledger systems in these three design alternatives had a "central management scheme" in all cases, managed either by the central bank or an intermediary, rather than a "distributed ledger scheme" where a single ledger is jointly managed by the participants in the transaction.

a fixed value, and the CBDC's holding status is indicated by linking those IDs to user IDs, called a "token-based CBDC ledger system."[2]

In all the ledger systems, the central bank issues CBDC to intermediaries, which is paid out to end users via those intermediaries. Paid-out CBDC is transferred between end users. Intermediaries accept CBDC from end users, and the central bank redeems it via intermediaries.

Figure 1: CBDC ledger design alternatives



Note 1: ①: Issuance; ②: Payout; ③,⑤: Transfer within a single intermediary institution; ④: Transfer across intermediaries; ⑥: Acceptance; ⑦: Redemption.
Note 2: In Design 2, payouts (②), transfers across intermediaries (④), and acceptance (⑥) simultaneously increase and decrease account balances on the central bank's ledger and on intermediaries' ledgers. Movements of transfers within a single intermediary institution (③,⑤) are not reflected in the central bank's ledger.

[2] Refer to Appendix 1 for characteristics of a token-based CBDC ledger system. Note that in PoC Phase 1, we are using the terms "token" and "token-based CBDC ledger system" as explained in the main text, but these can have a variety of meanings depending on the context.

Design 1: The central bank increases (or decreases) the account balances on its ledger for intermediaries through issuance (or redemption). Intermediaries decrease (or increase) their account balances through payouts (or acceptances) to end users, whose account balances are increased (or decreased). With transfers, the remitter end user's account balance is decreased, and the recipient end user's account balance is increased.

Design 2: The central bank increases (or decreases) the account balances on its ledger for intermediaries (own accounts) through issuance (or redemption). Intermediaries decrease (or increase) their account balances (own accounts) on the central bank's ledger through payouts (or acceptances), and increase (or decrease) the account balances at those intermediaries (aggregated user accounts). Also, they increase (or decrease) the end user's account balances on the intermediary's ledger. Through transfers across intermediaries, the account balance of an end user is decreased on the remitter intermediary's ledger, and the account balance of an end user is increased on the recipient intermediary's ledger. Also, the account balance of the remitter intermediary (aggregated user account) is decreased on the central bank's ledger, and the account balance of the recipient intermediary (aggregated user account) is increased on the central bank's ledger. Through transfers within a single intermediary, a remitter end user's account balance is decreased on the intermediary's ledger, and the recipient's account balance is increased on the intermediary's ledger.

Design 3: The central bank creates (or deletes) on its ledger tokens linked to an intermediary through issuance (or redemption). It changes the linkages between tokens and intermediaries or end users through payouts, transfers, or acceptances.

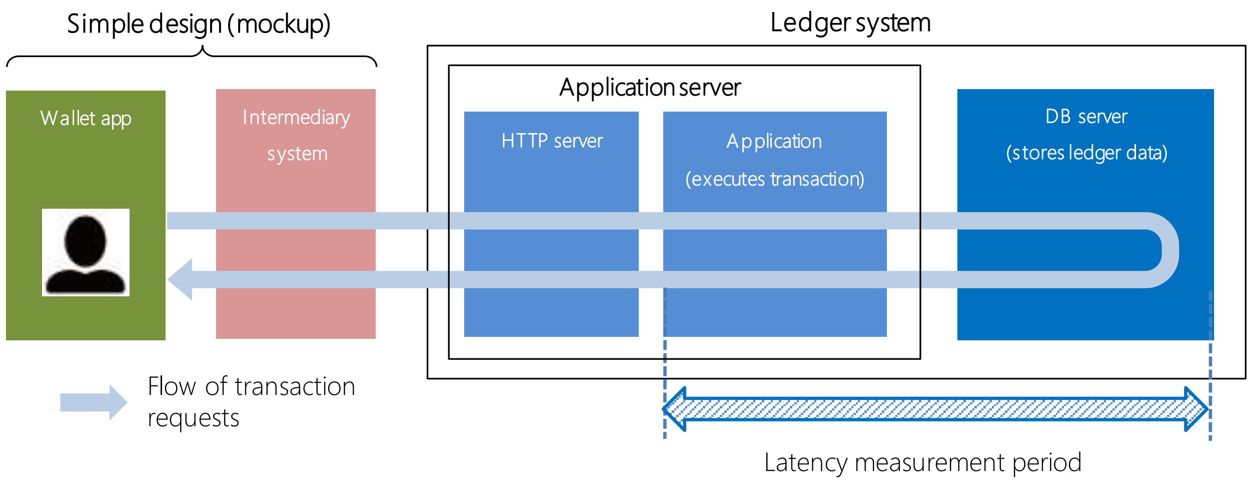
2.2 Experimental work[3]

Experimental environment

This experimental work is primarily intended to evaluate the processing performance of a ledger system handling the basic transactions of a CBDC. To accomplish this, we built an experimental environment in a public cloud (Figure 2). Here, the CBDC ledger system includes application servers that execute the transaction requests and a database server that records and retains the resulting holding status. The ledger system in Designs 1 and 3 is only for the central bank, while in Design 2, there are separate ledger systems for the intermediaries. External to the ledgers are the intermediaries' systems and the wallet apps used by end users; they have a simple design (mockup) that allows only the injection of transaction requests.

[3] For details on the system used in the experimental environment and methods for measuring processing performance, refer to Appendix 2.

Figure 2: Overview of the experimental environment


Simple design (mockup)
Ledger system
Application server
Wallet app
Intermediary system
HTTP server
Application (executes transaction)
DB server (stores ledger data)
Flow of transaction requests
Latency measurement period

Configuration

All three designs assume 100,000 end users and 5 intermediaries (2 large, 1 mid-sized, 2 small), with the intermediaries (mockups) injecting a number of transaction requests proportional to their scale. The breakdown of transaction requests by type is 5% payouts, 30% transfers within a single intermediary, 60% transfers across intermediaries, and 5% acceptances.[4] In addition, in Designs 1 and 2, each end user has their own account, and in Design 3, the total number of issued tokens is 25 million and in a single iteration of a transaction request, the number of tokens that will be updated is 10.[5]

Test scenarios

For PoC Phase 1, we estimated the processing performance required in introducing a CBDC in the future (production-ready system): typical throughput of several tens of thousands of transactions per second, peaks beyond 100,000 transactions per second, and latency of within a few seconds, and towards that end created a test scenario intended to reveal performance

[4] In addition, the rate of issuance/redemption transaction requests between the intermediaries and the central bank was set to one per minute per intermediary. In a production-ready system, it is assumed that balance inquiries would inject a reasonable number of transaction requests, but because this experiment focused on evaluating the performance of payouts, transfers, and acceptances in particular, we set this as one per minute per intermediary.

[5] In Design 3, the need to be able to handle cases where the amount transferred by an end user does not match any group of held tokens means that the intermediary will make exchanges with end users using the tokens the intermediary has on hand. Based on preliminary simulations, the number of tokens to be updated in one iteration of transaction requests was tentatively set to 10, and 50% of transaction requests were tentatively assumed to involve these exchanges. Based on these tentative settings, a 15-minute performance test was run during which transfers and exchanges were repeated. To avoid depleting the number of tokens held by end users and intermediaries, the test was run with a starting condition of each end user holding 200 tokens and the intermediaries holding 5 million tokens across the five of them; thus, the total number of outstanding tokens was 25 million. (Refer to Appendix 1 for characteristics of a token-based CBDC ledger system.)

differences and bottlenecks between the design alternatives. Specifically, for each of the designs, we created a "base scenario" in which 500 transaction requests are injected over a second, being a level each design should be able to handle without any delay of process, and a "high-load scenario" in which 3,000 transaction requests are injected over a second, being a level that should affect each design's processing performance.

Measurements

The following three measurements evaluate each design's processing performance.

(1) Throughput: the number of transaction requests that a CBDC ledger system could process per second (the number of transactions processed successfully per second by the application server's application).

(2) Latency: the time taken to process one transaction request (the sum of application and database processing times).

(3) Resource utilization: CPU utilization rates for the application server and database server.

2.3 Architecture evaluation

In architecture evaluation, we evaluated the "performance expandability" of each design. Specifically, based on the results of performance evaluation in experimental work, we examined the strategies to achieve the processing performance in a production-ready system for each design, and the issues that would be faced.

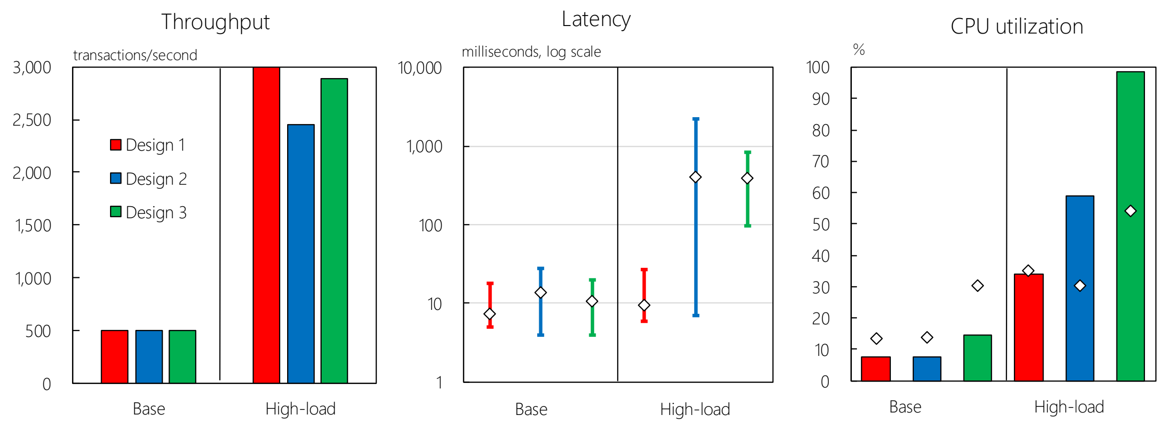
In addition, we compared and evaluated the reliability (resistance to security risks, fault tolerance, and availability) and ease of extension (ease of implementing additional functions) for each design of CBDC ledger systems.

3. Main results

3.1 Results of performance evaluation

Under the base scenario, each of the designs achieved a throughput of 500 transactions per second, which was the target we had set beforehand, a latency of roughly 10 milliseconds, and CPU utilization rates of 10 to 30% (Figure 3). Under the high-load scenario, Design 1 achieved a throughput of 3,000 transactions per second, which was equivalent to the target rate set for transaction requests, and latency was mostly unchanged from the base scenario. Designs 2 and 3, however, saw the throughput drop by 18% and 4%, respectively, relative to the transaction requests, while latency at the 99th percentile increased significantly to 2,000 milliseconds and 800 milliseconds, respectively.[6] The utilization rates of the application server and database server were higher for all designs under the high-load scenario than the base scenario, and in Design 3, the database server's CPU utilization rate approached the upper bound of 100%.

Figure 3: Performance evaluation results



Note: Figures are average values for the measurement period. Latency shows ranges from the 1st to 99th percentiles,with averages indicated by ◇. CPU utilization is for the central bank's ledger, with bars indicating values for the database server and ◇ the application server.

Below, we describe the source of the bottlenecks leading to performance declines in Designs 2 and 3.

Design 2 performance bottlenecks

Looking in detail at the processing status of the database server in Design 2, we saw numerous occurrences of "concentration of processing in account-balance data" that we did not see in Designs 1 and 3. In the experimental environment we built, if there are multiple transaction requests that update account-balance data (records), records are locked so that a later transaction cannot be processed until the earlier transaction is complete (record locking). When the earlier process is complete, the record lock is lifted and the later process is executed. In Design 2, especially with requests for transfers across intermediaries (Figure 1, Step 4), processing will be concentrated around the aggregated user account at the corresponding

[6] Latency in the figure shows the sum of application and database server processing times, but looking at this in more detail, almost all of the increase takes place in database server processing times.

intermediaries. The processing delay incurred by this record locking increases latency for Design 2 compared to Design 1.[7]

Record locking is one of the factors that led to the increased CPU utilization rate for the database server under the high-load scenario. In short, during the time that a record is locked, processing is paused on the database server, and the status of the record needs to be monitored constantly so that the later transaction is processed as soon as the record lock is lifted. This means that the CPU utilization rate increases with the number of processes on hold for a single record.

Design 3 performance bottlenecks

Because the database server in Design 3 had a CPU utilization rate approaching 100% under the high-load scenario, we see that processing exhausted the available resources. The main factor behind this is that for each transaction instruction, holder ID updates were processed for multiple tokens, and because exchange processes happen for a constant fraction, the number of processes ballooned compared with the other designs. These resource constraints decreased throughput and increased latency.

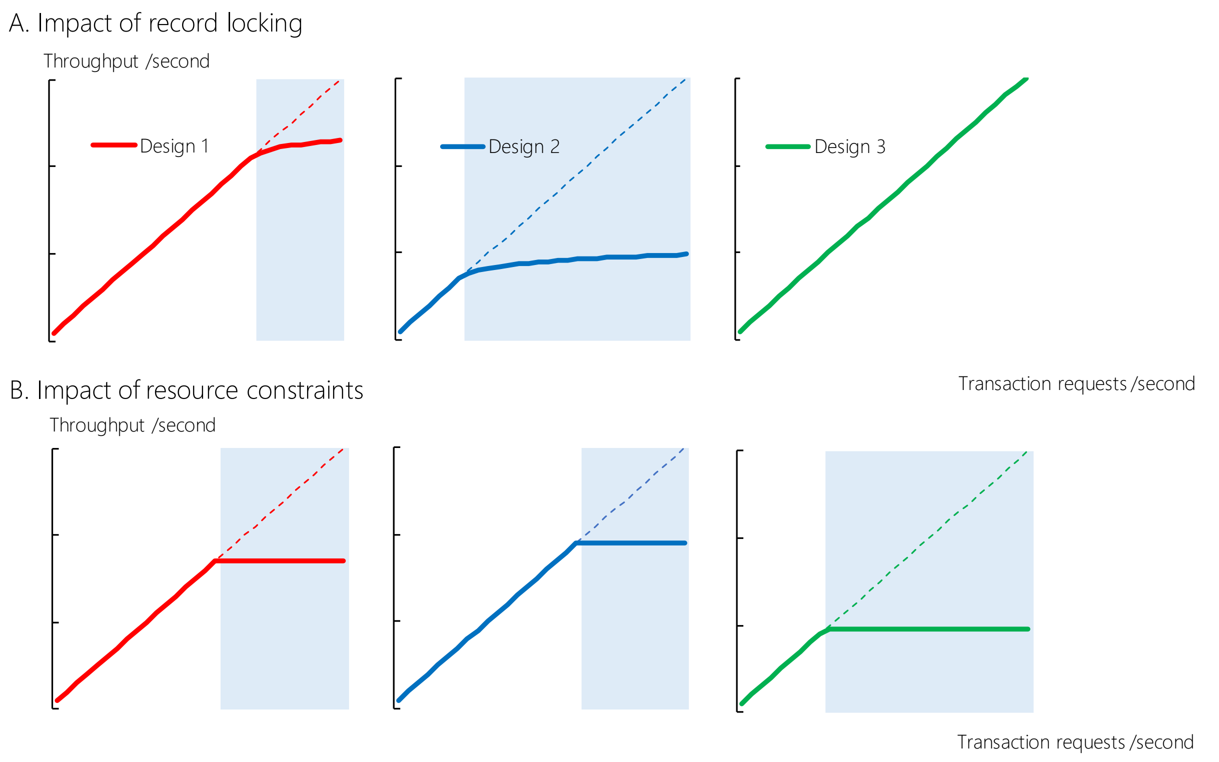
3.2 Performance issues towards achieving a production-ready system

Based on the results of the performance evaluation reported in Section 3.1 above, we examined the strategies to achieve the processing performance needed towards achieving a production-ready system and the issues that would be faced.

The performance evaluation results showed that the record locking in Design 2 and the resource deficiency in Design 3 each brought about performance bottlenecks (Figure 4). Record locking did not pose a problem in Design 1, but it could arise in an account-based CBDC ledger system depending on the number of transaction requests. Similarly, the impact of resource constraints was most evident in Design 3, but a sufficient increase in the number of transaction requests could manifest this problem in Designs 1 and 2 as well. Coming up with solutions that remove these bottlenecks is crucial if we are considering a production-ready system.

[7] For details on the impact of record locking, see Appendix 3.

Figure 4: Relationship between the number of transaction requests and throughput



Note 1: The graphs approximate the relationship between transaction requests and throughput. Design 2 is for the central bank's ledger.

Note 2: The shaded areas indicate where a bottleneck is causing throughput to decline relative to the number of transaction requests. The dotted lines show the extrapolated curve without the bottleneck.

Note 3: All graphs for A and B exclude any reciprocal effects.

One possible strategy for mitigating the impact of record locking is "record splitting," where the account-balance data recorded in the CBDC ledger are split into multiple parts. For example, in Design 2, splitting the "aggregated user account" account-balance data in the intermediaries by dividing end users affiliated with an intermediary into groups would reduce the processing concentration around an aggregated user account.[8] Another option is to redesign the business process flow. For example, in Design 2, processing requests for CBDC transfers across intermediaries (Figure 1, Step 4) are currently designed as simultaneous account transfers on the central bank's ledger and on the intermediary's ledger, but relaxing this simultaneity requirement and allowing the two transfers to take place separately can be considered to mitigate the impact of record locking, at least in a technical sense.

[8] Note that if account-balance records are split into too many groups, the number of records that would need to be added up for calculating account balances would also increase, possibly resulting in longer processing times. As such, it should be kept in mind that processing performance could decline depending on such factors as the record-splitting technique.

There are two ways to tackle the problem of constrained resources: scaling up (increasing a server's processing performance) and scaling out (adding more servers). Assuming record locking are solved, we can calculate that relying on scaling out alone to solve resource constraints in a production-ready system (assuming a throughput of 100,000 transactions per second) would require approximately the same number of database servers (for the central bank's ledger) for Designs 1 and 2, but two to three times as many for Design 3. When increasing the number of database servers, it is necessary to carefully optimize the design and layout of the databases while bearing in mind that individual database servers manage different records, so as to avoid deterioration in throughput and latency due to transaction requests being concentrated in certain records.

In a production-ready system, there could be bottlenecks outside the scope of the PoC (network bandwidth, storage I/O performance, etc.). There could also be scenarios unlike the ones we tested, such as temporary and localized extreme loads (for example, when a very large remittance is made to a certain end user or when one type of transaction request makes up an unusually high proportion of the total). Additionally, it would become necessary to meet additional needs such as implementing increasingly complex additional functions and sufficiently dealing with security risks. A production-ready system would need to fully account for all these factors in order to design and build a suitable system.

3.3 Non-performance issues towards achieving a production-ready system

Reliability

The reliability a CBDC ledger system would require has three aspects: resistance to security risks, fault tolerance, and availability. We compared each of the designs in terms of each these aspects. We confirmed that in terms of fault tolerance particularly, Designs 1 and 3 were different from Design 2, but each had its strengths and weaknesses, and there were a fair number of issues they all had in common that would need to be solved in a production-ready system. We discuss these below.

First, regarding resistance to security risks (resistance to cyberattacks), in order to perform a generalized assessment that was not limited to any specific security threat, we elaborated the typical threats, such as wiretaps and information tampering, in addition to DDoS and other cyberattacks, and investigated the strategies that would be needed to counter these. Our investigations showed no major differences between the design alternatives at the system level. In short, the constituent elements of a CBDC ledger—the application servers and database servers, along with the networks between them—each would need their own specific security strategies, which would be the same regardless of the design. Note that in Design 2, which splits ledger management, both the intermediaries and the central bank need to have appropriate security strategies in place for the CBDC ledger.

In order for a CBDC to be secure, an adequate level of security would need to be assured regardless of the design, but this could make system processing performance lower or increase deployment/operating costs. To implement a production-ready system and build a stable efficient settlement system, these tradeoffs would need to be carefully balanced.

Second, regarding fault tolerance (the number of potential failure points and the scope of their impact), there are differences between design alternatives that depend on the differences in ledger management. For example, with Design 2, even if a fault occurs in a ledger managed by the central bank, a CBDC transfer between end users having accounts at the same intermediary would work, and the expected impact area would be relatively small. Conversely, because the ledger is split between the central bank and each intermediary, there are more potential points of failure than in Designs 1 and 3. Also, we anticipate that the integrity of restored data (reconciling changes in each ledger's balances or deposits) is unlikely to hold.

Implementing a production-ready system requires an investigation into the fault tolerance of not just the CBDC ledger but the CBDC system as a whole. For example, in both Designs 1 and 3, there are parts where each intermediary handles operations and management of the system, for example, for providing the interface to end users and exchanging messages with the central bank. In terms of the differences between designs, production use will require a holistic assessment that takes these points into consideration.

Third, regarding availability (instances of system downtime and their duration), we do not see major differences between the design alternatives. That is, preparing for a fault is assumed to require building multiple sites in advance, and the ease of doing that would be the same regardless of the design. Note that in Design 2, since each intermediary keeps its own copy of the ledger in addition to the central bank, each intermediary would need to develop strategies to improve the availability of its own ledger.

To implement a production-ready system, common considerations for all of the design alternatives would be avoiding or minimizing lag in failover during a fault and accommodating planned outages for system maintenance. If a CBDC ledger system needs to have high availability, a "one-site/two-systems" configuration or a distributed-load configuration using multiple sites will be required, but these would result in additional setup and operating costs, which would need to be balanced against other priorities.

Ease of extension

In PoC Phase 1, we ran experimental work of basic CBDC transactions, and towards achieving a production-ready system, any number of additional functions would be implemented. On that point, we undertook architecture evaluation of the issues that would be faced in implementing these additional functions. As a result, we confirmed that in Design 2, in order to implement additional functions that assume mutual lookups and totals of CBDC holdings recorded in the ledgers of different intermediaries, for example, we would need to build additional processes to perform those. In Design 3, when we considered an additional function that assumes acquiring each end user's CBDC holdings in every transaction, we came to the conclusion that we would need to build an additional function to calculate the total amount of tokens held by each user.

As an example of "limits on CBDC holdings," consider the following. It would be possible to deploy a function that would check whether a transfer request resulted in a collision with some upper limit on the recipient's holdings whenever a CBDC transfer request is executed. In Design 2, this would require an additional process that would link the balance information between the ledger systems of different intermediaries. The reason for this is that ledgers are distributed among multiple intermediaries in Design 2, and when executing transfer requests across intermediaries, we expect that the remitter's intermediary would check whether the transfer would result in a collision with the recipient's upper limit on CBDC holdings at the recipient's intermediary. In the token-based Design 3, the ledger model only records links between individual tokens and end users, and an individual end user's total CBDC holdings are not recorded anywhere. For this reason, with each transfer request, there would need to be a calculation to check whether the recipient's CBDC holdings had collided with this upper limit.

In this way, we see that Designs 2 and 3 are at a disadvantage to Design 1 in terms of ease of implementing additional functions. Still, even Design 1 could implement functions for linking information between intermediaries as mentioned above to perform a counterparty check in a CBDC transfer. Therefore, this would not be a problem unique to Design 2. Design 2 brings a greater degree of freedom to ledger design at intermediaries, so this could have advantages in terms of building a CBDC ledger that suits the specifications of a core banking system, or developing additional services in line with end-user needs. With Design 3 as well, we can imagine additional functions and services that would work by appending additional information to tokens. Thus, each design may offer advantages.

Considering all of the above, while each design has different characteristics, we cannot go so far as to say that any of them stand out in terms of ease of extension. Additional functions that might be implemented in the future could take the forms of "limits on CBDC holdings," "booking remittance transaction from end users," "limits on CBDC transaction amounts," etc. PoC Phase 2 and what follows will need to evaluate the technical feasibility of implementing these additional functions and their impact on processing performance in more detail.

4. Key findings and next steps

In PoC Phase 1, we compared and evaluated design alternatives to determine whether they could accurately process basic transactions related to CBDC, on the assumption that in a production-ready system, processing performance would need to be on the order of tens of thousands of transactions per second typically, and 100,000 or more at peak, with latency of within a few seconds.

The results of our experimental work showed that processing perform lower in Design 2 compared with Design 1 due to the effects of record locking. That said, Design 1, which is also based on an account-based ledger system, could experience the similar lower performance depending on the number of transaction requests. In order to build a production-ready system, we would need to consider solutions such as split records and different business process flows to address these issues.

Design 3 required more resources than Designs 1 and 2 to process the same transactions. Designs 1 and 2 will be constrained by resources if the number of transaction requests is up significantly. In a production-ready system, Design 3 in particular would need greatly expanded resources, either through scaling up or scaling out of database servers, with the additional optimizations.

Apart from these performance considerations, we also looked at resistance to security risks and availability, but did not see any significant differences arising from the variation in the designs. In terms of fault tolerance, Design 2 seems to have a relatively small expected impact area for faults, but at the same time there are many potential points of failure, and the integrity of restored data is unlikely to be hold. Regarding ease of extension, each design has its distinct qualities, but whether major differences exist among them remains yet to be determined.

The above findings and issues arising from the differences among the design alternatives may change as needed functions are added to the CBDC ledger system or as systems outside the ledger are built. For this reason, in PoC Phase 2 we find it appropriate to explore in more depth technical considerations involved in a production-ready system. In doing so, we will build on the CBDC ledger systems developed in Phase 1 more complex additional functions and explore their technical feasibility as well as their possible impact on processing performance.

Appendix 1: A token-based ledger system

In a token-based ledger system, fixed-value monetary data (tokens) are linked to holders. When processing payouts, acceptances, and transfers, the holders linked to the tokens are changed. In a token-based ledger system, if a remitter holds a sufficient number of tokens, there is a low probability of concentrated record processing as in an account-based ledger system, and it rarely happens processing delays caused by waits for record locks to be lifted. Conversely, a single transaction request will require that many tokens be updated, and to that extent, resource utilization will increase.

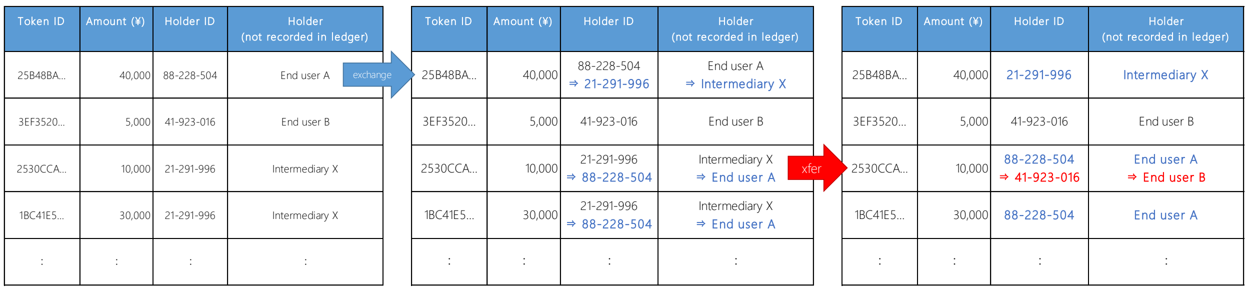
In a token-based ledger system, apart from the "fixed-value approach" in which the user linked to an existing token is changed in a CBDC transfer, there is also the "flexible-value approach," for example, in which tokens are split or merged, and the newly generated tokens are linked to end users in a CBDC transfer. Design 3 used in PoC Phase 1 takes the fixed-value approach. The following is a brief description of the fixed-value and flexible-value approaches.

Fixed-value approach

In the fixed-value approach, if the end user does not hold a set of tokens with a value that matches the transfer amount, the intermediary performs an exchange so that the end user can obtain smaller-denominated tokens, and these are then transferred. There might also be the option for getting "change" from the recipient, but in this case, the recipient would need to have a set of tokens with a value that matches the "change" amount, so additional considerations in business process flow would be needed.

Unlike our current experiment, which presupposes online settlements, if offline settlements is in prospect, in which a pair of end users use their endpoint devices to communicate in order to transfer CBDC directly, with the information being stored independently on those devices, a critical question becomes whether the tokens being used are authentic. On that point, because the tokens used in the process have a fixed value, having not been split or merged, each token can carry the electronic signature that the central bank would apply to each token upon issuance, and because that would remain unchanged when transferred, if the end users verify the electronic signatures, the authenticity of the tokens can be effectively confirmed.

Appendix figure 1-1: Transferring JPY 10,000 from A to B in a fixed-value approach


exchange
xfer

Token ID	Amount (¥)	Holder ID	Holder (not recorded in ledger)
25B48BA...	40,000	88-228-504	End user A
3EF3520...	5,000	41-923-016	End user B
2530CCA...	10,000	21-291-996	Intermediary X
1BC41E5...	30,000	21-291-996	Intermediary X
:	:	:	:

Token ID	Amount (¥)	Holder ID	Holder (not recorded in ledger)
25B48BA...	40,000	88-228-504 ⇒ 21-291-996	End user A ⇒ Intermediary X
3EF3520...	5,000	41-923-016	End user B
2530CCA...	10,000	21-291-996 ⇒ 88-228-504	Intermediary X ⇒ End user A
1BC41E5...	30,000	21-291-996 ⇒ 88-228-504	Intermediary X ⇒ End user A
:	:	:	:

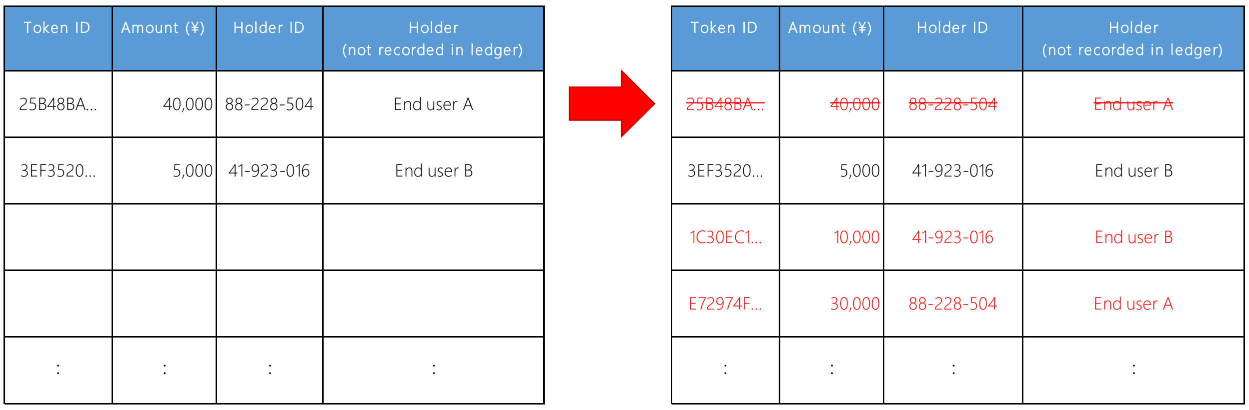
Token ID	Amount (¥)	Holder ID	Holder (not recorded in ledger)
25B48BA...	40,000	21-291-996	Intermediary X
3EF3520...	5,000	41-923-016	End user B
2530CCA...	10,000	88-228-504 ⇒ 41-923-016	End user A ⇒ End user B
1BC41E5...	30,000	88-228-504	End user A
:	:	:	:

Flexible-value approach

In the flexible-value approach, if the end user does not hold a set of tokens with a value that matches the transfer amount, high-value tokens are split into low-value tokens. When this happens, the token is deleted before the split, and after the split, tokens with the value to be transferred are linked to the recipient, and the remaining tokens are linked to the remitter.

To the extent that the flexible-value approach does not require an exchange process, it has shorter processing times for CBDC transfers than the fixed-value approach, and by periodically merging tokens (concentrating sets of low-value tokens into a high-value token), the size of the database can be compressed. Conversely, when we consider offline settlements, there needs to be a way to confirm the authenticity of tokens that are going to be split on the end-user side, so this will require further study.

Appendix figure 1-2: Transferring JPY 10,000 from A to B in a flexible-value approach



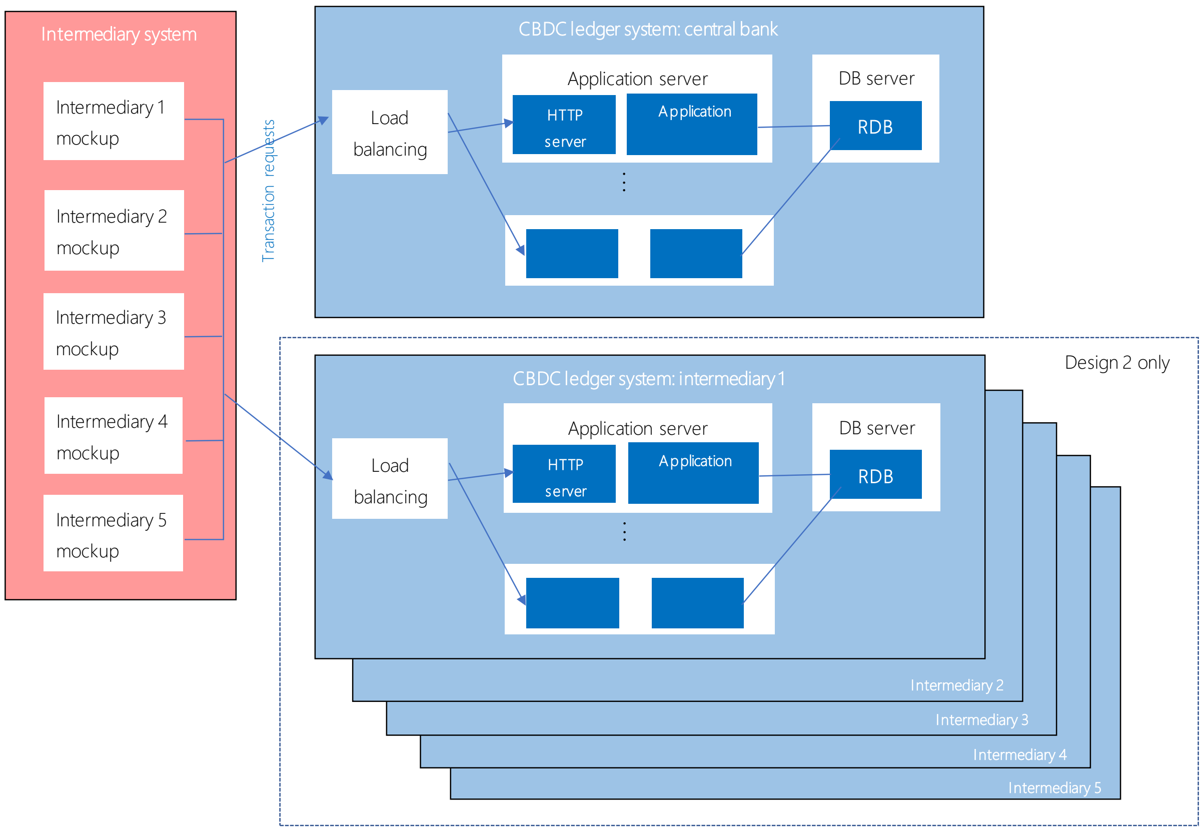
Token ID	Amount (¥)	Holder ID	Holder (not recorded in ledger)
25B48BA...	40,000	88-228-504	End user A
3EF3520...	5,000	41-923-016	End user B
:	:	:	:

Token ID	Amount (¥)	Holder ID	Holder (not recorded in ledger)
~~25B48BA...~~	~~40,000~~	~~88-228-504~~	~~End user A~~
3EF3520...	5,000	41-923-016	End user B
1C30EC1...	10,000	41-923-016	End user B
E72974F...	30,000	88-228-504	End user A
:	:	:	:

Appendix 2: Experimental environment and measurements of processing performance

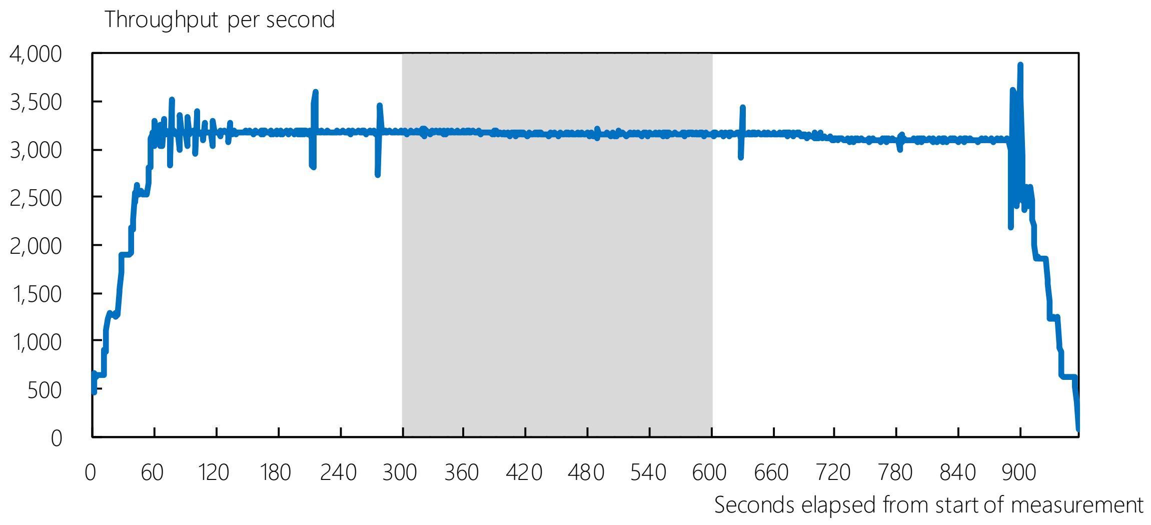
The CBDC ledger system in our experimental environment had two major components: an application server that executed the transaction requests and a database server (relational database) that recorded and retained the results. Each server has a vCPU with 8 cores, 64 GB memory, and SSD storage; under the base scenario tests, we used two application servers, and under the high-load scenario tests for each design, we added more application servers until resource constraints were removed. We used one database server for each design in both the base and high-load scenario tests. In Design 2, in addition to the central bank, each intermediary controlled the same sort of CBDC ledger system. The intermediary systems were just mockups that let us inject transaction requests from outside.

Appendix figure 2-1: Experimental environment diagram



For the performance evaluation in our experimental work, we injected transaction requests for a period of at least 15 minutes under both the base and high-load scenarios (500/second and 3,000/second, respectively), and treated the period from minute 5 to minute 10 as subject to evaluation, in order to obtain stable measurements.

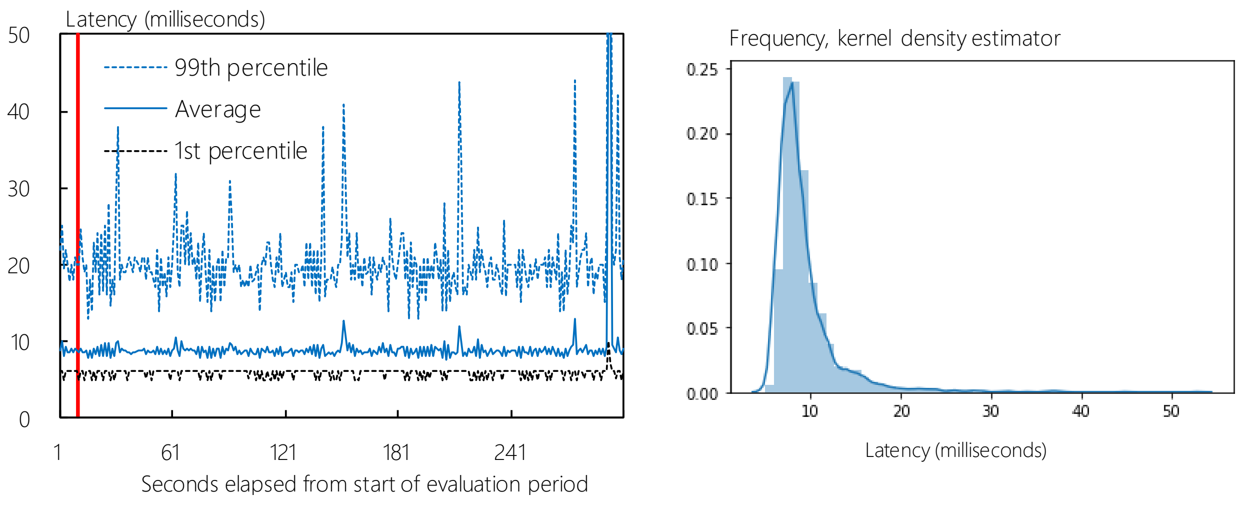
Appendix figure 2-2: Throughput under high-load scenario for Design 1



Shaded area: evaluation period

A histogram of latency—the time required to process a single transaction request—will show most measurements concentrated near the average, but with occasional extremely high measurements, resulting in a "long-tail" distribution. In addition to the average value during the evaluation period, we also measured the 1st percentile and 99th percentile to reflect this.

Appendix figure 2-3: Latency under high-load scenario for Design 1

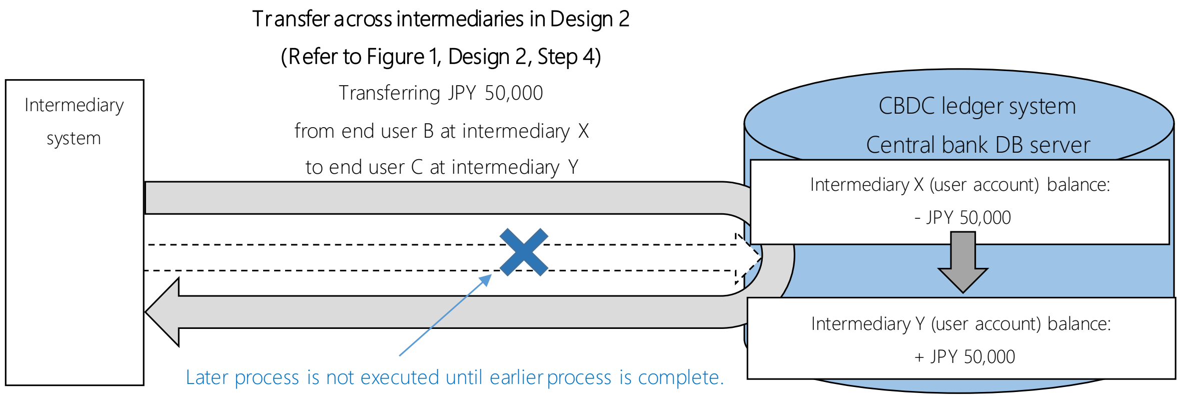


Note: The right-hand graph shows latency with respect to the number of transaction requests at 20 seconds after the start of the evaluation period (red line in the left-hand graph).

Appendix 3: Impact of record locking

With account-based CBDC ledger systems, as used in Designs 1 and 2, when there are multiple transaction requests performed to update account-balance data (records), the records being processed—the remitter's account-balance data and the recipient's account-balance data—are locked, so that a later process cannot act on those data before an earlier process has completed. This is record locking. A hypothetical system without record locking would allow multiple database updates simultaneously, but this would lead to incorrect balances management.

Appendix figure 3: Record-locking schematic


Transfer across intermediaries in Design 2
(Refer to Figure 1, Design 2, Step 4)
Transferring JPY 50,000
from end user B at intermediary X
to end user C at intermediary Y
Intermediary system
CBDC ledger system
Central bank DB server
Intermediary X (user account) balance:
- JPY 50,000
Intermediary Y (user account) balance:
+ JPY 50,000
Later process is not executed until earlier process is complete.

As this shows, record locking is needed to ensure the correct sequence in settlements, but when multiple processes that need to act on the same record take place, they adversely affect system processing performance. Particularly in Design 2, because end-user transaction statuses and balance fluctuations all are reflected in the intermediary accounts (aggregated user accounts) on the central bank's ledger, requests for transfers across intermediaries tend to be concentrated around those accounts, and therefore processing is delayed while waiting for record locks to be lifted.

A technique for reducing the impact of record locking that has been considered is transaction splitting, in addition to the record splitting and business process flow redesign mentioned in Section 3.2 of the main text. In this, a CBDC transfer request is split into two parts: a process that reduces the remitter's account balance (withdrawal) and a process that increases the recipient's account balance (deposit). These processes can run independently, and although the total number of processes being run on the system increases, the congestion caused by multiple processes trying to act on the same record is alleviated. In our experiment in Design 2, we had the bare minimum of record splitting and transaction splitting in place so that the base scenario would run smoothly at 500 transaction requests per second.

While transaction splitting has the effect of reducing the impact of record locking, it does make it possible for data to get out of sync. Because it can be a factor in dumping the integrity of a single transaction request's process, the pros and cons would need to be weighed carefully before using transaction splitting in a production-ready system.