An Adaptive System for a Real-Time Matching Application

Taka Matsutsuka, Fujitsu Limited, Japan
Masatoshi Ogawa, Fujitsu Limited, Japan
Yohei Toriyama, Fujitsu Limited, Japan
Noriyasu Aso, Fujitsu Limited, Japan
Ichiro Iida, Akita Prefectural University, Japan

ABSTRACT

In order to enhance the customer experience, it is important not only to provide functions, but also to respond to changes in environments and requirements. It is a difficult task to evaluate and manage which function with different locations and contents is most valuable to the user’s experience without using computationally time-consuming optimization calculations. To address this, this paper is focusing on self-adaptive software technology. The authors built a software adaptation mechanism that can be immediately calculated online using a performance characteristic map and threshold judgment with a learning function and sequential updates. The results confirmed the effectiveness of the mechanism in an application that supports personnel exchange events.

KEYWORDS

Matching Application, Mobile Application, Personnel Exchange, Self-Adaptation Software, Web Application, Web Service

INTRODUCTION

In recent years, companies are experiencing more uncertainty than before due to increasingly severe changes in the business environment. In such a situation, efforts focusing on maximizing customer experience (CX) draw attention in the industry (Webb, 2019). In order to enhance CX, it is important not only to provide functions, but also to respond to changes in environments and requirements. To address this in software systems, technology is needed to dynamically change the behaviour of the system in response to changes in runtime. Self-adaptive software technology is being actively researched to address this type of issue (Weyns, 2017). In particular, the authors are focusing on improving user experience through a self-adaptive mechanism for web applications running in the cloud.

The proliferation of HTML5 and the improvement of client device performance have allowed execution of various processes on the client side of Web applications. Now, the client side can execute functions which have been previously executed on the server side (Hales, 2012). While this can reduce the load on the server and increase the response to the users’ input, from the viewpoint of enhancing performance, it is not always effective to execute the processing on the client side (Walker & Chapra, 2014). On the server side, the performance depends on a number of factors, such as the number of client devices currently connected and load fluctuations due to the execution of other applications on the same server, which can change from moment to moment (Sharifian, Motamedi, & Akbari, 2011). In this situation, the applications need to determine whether it is most appropriate for the process to be executed on the server or the client without adding significant overhead. Furthermore, a function may not be completed in a meaningful time frame, regardless of whether it is executed on
the server or the client, due to sudden environmental changes. In order to cope with such exceptional circumstances, it is necessary to provide a mechanism for switching to a fallback function to prevent CX to degrade (Tomás, Masoumzadeh, & Hlavac, 2016).

In this research, we focus on an application for personnel exchange event, which is required to immediately return the processing results to the user. We consider a mechanism that can autonomously decide in a very short time whether to execute (a) a function on the client, (b) a function on the server, or (c) a fallback (simplified) function because the nature of the event is greatly affected by changes in the environment such as the number of participants and the network environment of the venue. Existing technologies addressing this type of problem includes model-based optimization and control (Cailliau & Van Lamsweerde, 2017; Moreno, Camara, Garlan, & Schmerl, 2018; Shevtsov, Weyns, & Maggio, 2019). However, these methods require relatively long computation time. Cyber Foraging (Mahadev, 2001) is an effective method to speed up model-based optimization by having a nearby powerful server take over the computation-intensive processing, but the problem is that the prediction model needs to be retuned when the environment changes.

Therefore, we propose an adaptation mechanism that can be computed immediately online by applying a mapping table with learning capabilities, its update rules, and threshold rules, instead of using sequential optimization computations. This allows the system to autonomously evaluate and manage whether it is most valuable for the user’s experience to execute (a) a function on the client, (b) a function on the server, or (c) a fallback function with reduced functionality but guaranteed to be done in short time. The mapping table can learn incrementally as the environment changes, and the threshold decision is robust to rapid environmental changes. In addition, the mapping table can also learn the results from other users’ runs to reflect continuous changes in the environment.

This paper is composed as follows: Application and Its Challenges section introduces the target application; in Method section, the authors propose a mechanism to deal with issues; in Evaluation section, the proposal is evaluated against the application introduced in Application and Its Challenges section; in Related Works section, the authors discuss related studies and implications; and lastly Conclusion section draws a conclusion.

APPLICATION AND ITS CHALLENGES

Target Application

In the field of innovation management, deepening expertise and obtaining diversity are both important for innovation creation (O’Reilly & Tushman, 2016). To facilitate innovation, we need the so-called “cross-functional team,” which consists of people with diverse expertise working on a common goal. In order to build a cross-functional team, personnel exchange events such as meetups and workshops are held to find the necessary personnel.

Buddyup! (Fujitsu, 2021) is an application that is used to find the desired people for each other in a personnel exchange event. When a participant registers a self-introduction text in the application, the text is analyzed morphologically and the analyzed words become candidates for words related to skills and interests (hereafter referred to as tags). The tag candidates, which may include words not related to the skills and interests, will be checked against the words in the Tag DB and, if they match, will be published as tags. Recommending other participants who share common tags makes it easier for the user to find people who share the same interests as him/her. This eliminates the need to repeatedly ask questions at the event to elicit information about the other person and facilitates the exchange of talent.

Buddyup! is implemented as a Web application (Figure 1). Users use a Web browser on their smartphones and other devices. Buddyup! executes most of the functions on the client device. The server application is deployed in the cloud to manage data for each event. The server also has an
engine for analyzing a self-introduction text and a database for tags to be analyzed. The client calls the server when it needs, for example, to access database or analyze self-introduction.

In this study, the authors adopted Buddyup! to increase CX. At a personnel exchange event, the participants need to find people to communicate with each other in a limited time frame and to extract other people’s information from the absence of mutual understanding. To do this, the application extracts tags from the participants’ self-introductions and matches the tags between the participants. The application returns matching results in a timely manner while the self-introduction is being edited during the event. This is a unique distinction from other matching applications where the matching process can be done over time using batch processing (Wang, Hu, Zhang, & Cao, 2017).

**Application Challenges**

In the personnel exchange event, strangers meet and match each other on the spot, based on the input of their self-introductions. Since the self-introduction changes during the event, the tag matching process for participant recommendation should be done in real time, and the results should be provided immediately.

Because the time required for the matching process depends on the participant's tag, Buddyup! implements two different matching algorithms with different computational complexity. Figure 2 shows an overview of matching:
• **nx1-matching:** The algorithm extracts \( n \) other participants (hereinafter refer to the number of matched persons \( n \)) having the same tag for every tag a user has. When a user has two or more tags, the tags are sorted in order of the number of extracted other users. The computational complexity of the function is \( O(n) \) as this process loops for \( n \) users.

• **nxm-matching:** For every \( m \) tags a user has, the algorithm extracts \( n \) other participants with the same tags in common. From the results of the nx1-matching, the algorithm forms groups of 2-5 people and extracts the tags that all match. For example, in the case of a group of 3 people (i.e. for \( C^3_n \) patterns), the algorithm extracts the tags that are common to 3 people in the same group. It processes the same for groups of 2, 3, 4 and 5 people. The result starts from groups with the most \( m \) common tags. For the same number of common tags, the result is in order of the number \( n \) in the group. The computational complexity of the function is \( O(n^4) \) as the algorithm needs to process the groups of 4 sets of combinations.

Since nxm-matching can provide the richer result than nx1-matching, it can provide higher CX. On the other hand, as the number of users increases, the calculation time and the waiting time increases, resulting in decreased CX. Since the calculation time depended on the number of matched persons \( (n) \), nxm-matching was used only if the number of participants was expected to be below a certain threshold.

However, from the viewpoint of CX, it is better to execute nxm-matching as much as possible. The authors attempted to compare the nxm-matching process between on the server and on the client. When the number of matched persons was less than a certain number, the client was faster, otherwise the server was faster. This means that nxm-matching should be executed by the client when the number of matched persons is small and by the server when the the number is large, to obtain the best result. However, the challenge consists in the following:

- Since the load of the client and the server changes constantly, the optimum number of matched persons to switch between the server and the client varies depending on the load condition.
• Since the client device is different for each user, the optimum number also varies depending on the client device.
• Even in the case of server processing, the processing time increases as the number of matched persons increases. In order to maintain CX, the absolute value of the response time should be less than a certain value, so it is necessary to fallback to nx1-matching at a certain threshold.

In the next section, the authors describe the method for addressing the above issues.

METHOD

Overview

When the application has alternative functions, such as a function to be processed on the client, a function to be processed on the server or a fallback function to secure the response time, which function to choose depends on the environmental conditions at the time. The environmental conditions here include, for example, a communication delay due to a congestion state of network between the server and the client, a delay of function processing due to a task load state on the client device, and an influence due to an increase or decrease in calculation load based on a change in input values of the function. Therefore, appropriate switching cannot be carried out by a static rule. This research aims to maximize the benefits of each user by selecting from multiple function candidates in an effective and adaptive way to respond to the changes in the environmental conditions of each user.

The Adaptation Module uses **Performance Characteristic Value (PCV)** to express the expected performance of target functions. PCV is the overall service performance value resulting from the execution of a function. This is what the result of executing the target function in the service module or server-side logic yields. It is derived from, for example, the response time at which the function returns a response result to the user, the cost of using the server, the accuracy of the output of the function, and the amount of information contained in the output of the function. The derivation formula is described later.

The Adaptation Module is composed of three submodules to implement adaptive operations. Figure 4 shows the structure:

- **PCV Management submodule** calculates, updates, and manages representative values of the performance of each function with respect to environmental conditions from the observed data, in order to respond to changing environmental conditions.
- **Control submodule** automatically generates a decision table capable of determining a function having the highest PCV for each environmental condition from the PCV of each function. Further, the observed latest PCVs and thresholds are used to respond to sudden environmental deterioration when the user’s waiting time becomes suddenly long.
- **Operation and Observation (O&O) Adapter submodule** asks the control submodule for the determination result of the function selection, selects the function based on the result, and executes the function. The PCV is calculated from the result of the function.
Figure 3. System Configuration

Frontend (Run on client device)

Adaptation Module
- PCV Mgmt.
- Control
- O&O Adapter

Service Module
- Display Mgmt.
- Other submodules are omitted

Target functions
- Function A
- Function B
- Function C
- ...

Backend

Database
- PCV database
- Service information

Logic
- Periodic Execution
- Relay Execution

Target functions
- Function A
- Function B
- Function C
- ...

Figure 4. Adaptation module

Adaptation Module

PCV Mgmt Submodule
- Update Tables

Control Submodule
- Switch Decision

O&O Adapter Module
- Call API
- Function Execution
- Measure

Service Module
- Read
- Require process
- Execute process

PCV Data notification

PCV

Table Data
ADAPTATION

Preoperational Processing

This subsection describes the preprocessing before the operation. An initial value of a PCV map for each function is set before the operation (see Table 1). The performance characteristic value (PCV) map is a mechanism that learns the performance of a function execution under a given environmental condition parameter. The PCV is updated sequentially each time the function is executed. Thus, it is possible to gradually adapt to changes in the environment. In order to reduce the amount of data held, the range of environmental condition parameters is divided into quantization slots and learning and referencing can be carried out on a quantization basis. An example of the PCV map is shown in Table 1. Here, a function identification number $n_{fid}$ identifies a given function. We define the moving average $\mu_{ma}$ of the PCV of the function, the moving variance $s_{ma}^2$ of the PCV of the function and the latest PCV $x$, respectively. In operation, the PCV $x^\left(n_{fid}, j, k\right)$ of the function identification number $n_{fid}$ is calculated for the quantization slot $j$ and the cycles $k$. The cycle is updated each time the function is executed. The moving average $\mu_{ma}^{\left(n_{fid}, j, k\right)}$ of the PCV $x^\left(n_{fid}, j, k\right)$ is calculated from that of the previous cycle $\mu_{ma}^{\left(n_{fid}, j, k-1\right)}$ and $x^\left(n_{fid}, j, k\right)$, the result is used to derive $s_{ma}^2^{\left(n_{fid}, j, k\right)}$. The detail of calculation will be given in PCV Management Submodule subsection. Here, $\mu_{ma}^{\left(n_{fid}, j, 0\right)}$, $s_{ma}^2^{\left(n_{fid}, j, 0\right)}$, and $x^\left(n_{fid}, j, 0\right)$ are set as initial values (i.e. $k = 0$) before the operation, using the results of desk evaluation.

For example, in the case of Buddyup!, nx1-matching result, indicating the number of matching users, is treated as an environmental condition parameter as the calculation cost of nxm-matching varies greatly depending on the number of matched persons in nx1-matching. The current moving average is used as a representative value of the PCV. The moving variance is used as an index for determining the degree of variation of the PCV at the time of threshold determination. The latest PCVs are used to respond sudden changes in threshold decision.

For each quantization slot $j$, a function identification number $n_{fid, opt}$ is derived as:

$$n_{fid, opt} = \arg \max_{n_{fid}} \mu_{ma}^{\left(n_{fid}, j, 0\right)}$$  \hspace{1cm} (1)

<table>
<thead>
<tr>
<th>Environment Condition Parameter</th>
<th>0-9</th>
<th>10-19</th>
<th>20-29</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantization slot No. $j$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>...</td>
</tr>
<tr>
<td>Moving Average</td>
<td>$\mu_{ma}^{\left(n_{fid}, 1, 0\right)}$</td>
<td>$\mu_{ma}^{\left(n_{fid}, 2, 0\right)}$</td>
<td>$\mu_{ma}^{\left(n_{fid}, 3, 0\right)}$</td>
<td>...</td>
</tr>
<tr>
<td>Moving Variance</td>
<td>$s_{ma}^2^{\left(n_{fid}, 1, 0\right)}$</td>
<td>$s_{ma}^2^{\left(n_{fid}, 2, 0\right)}$</td>
<td>$s_{ma}^2^{\left(n_{fid}, 3, 0\right)}$</td>
<td>...</td>
</tr>
<tr>
<td>Latest PCV</td>
<td>$x^\left(n_{fid}, 1, 0\right)$</td>
<td>$x^\left(n_{fid}, 2, 0\right)$</td>
<td>$x^\left(n_{fid}, 3, 0\right)$</td>
<td>...</td>
</tr>
</tbody>
</table>
from the initial value of the PCV map, and the decision table in Table 2 is created. The decision table makes it possible to derive the function identification number with the highest PCV for a given environmental condition parameter. The function identification number can then be used to execute the appropriate function for the current situation.

**Operational Processing**

**Table 2. Example of decision table**

<table>
<thead>
<tr>
<th>Environment Condition Parameter</th>
<th>0-9</th>
<th>10-19</th>
<th>20-29</th>
<th>30-39</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantization slot No. ( j )</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>...</td>
</tr>
<tr>
<td>Function Identification Number</td>
<td>( n_{fid, opt}(1) )</td>
<td>( n_{fid, opt}(2) )</td>
<td>( n_{fid, opt}(3) )</td>
<td>( n_{fid, opt}(4) )</td>
<td>...</td>
</tr>
</tbody>
</table>

The following subsection provides a description of the operation process shown in Figure 5.

**Figure 5. Flowchart in operation**

1. Start
2. Select a Candidate Function (CF) from the Decision Table
3. Calculate threshold (1)
4. The latest PCV \( \leq \) threshold (1)
   - Yes: Select a fallback function
   - No: Calculate threshold (2)
5. The latest PCV \( \leq \) threshold (2)
   - Yes: Select the function with the highest PCV under the condition where the environmental condition parameter is more severe than the present value.
   - No: Select CF
6. Execute selected function
7. Retrieve environmental condition parameter and sub-PCVs
8. Retrieve the results of other clients and the results of the periodic execution in backend
9. Calculate PCV from sub-PCVs
10. Loop for updating moving average, moving variance and the latest PCV for each function
    - \( t > N \)
    - Calculate moving average and moving variance for the function \( n_{opt}(i) \)
    - Update the PCV map for the function \( n_{opt}(i) \)
    - \( i = i + 1 \)
11. Loop for updating moving average, moving variance and the latest PCV for each function
    - Retrieve moving average, moving variance and the latest PCV corresponds to the current environment condition parameter from PCV map
12. Generate decision tables for each environment condition parameter
13. End
Control Submodule

When a service request from a user occurs, the Control submodule processes the following.

First, a candidate function (CF) corresponding to the environmental condition parameter is tentatively selected based on the decision table. If the expected time of the function is too long, it deteriorates user’s experience and the continuous use cannot be expected. The response time should be prioritized than the accuracy or the amount of the information provided to the user in such case. Threshold (1) $\theta_1(k)$ is the minimum PCV that all the functions required to provide. When the latest PCV of the CF for the current environmental condition parameter is $\theta_1(k)$ or less, the fallback function is selected to maintain user’s experience. A fallback function has a shorter response time, but the output of the function provides less precision and quantity of information to the user than the original function. The threshold (1) can be set in advance by using, for example, a hearing of a user.

When the latest PCV is more than $\theta_1(k)$, the submodule examines whether it is significantly lower than the variation of normal PCVs. The submodule calculates the threshold (2) $\theta_2(k)$ at the time of cycle $k$ from the moving average $\mu_{ma}(n_{j_0}, j, k-1)$ and the moving variance $s_{ma}^2(n_{j_0}, j, k-1)$ of the PCVs of the quantization slot $j$ corresponding to the same environmental condition parameter of the CF:

$$\theta_2(k) = \mu_{ma}(n_{j_0}, j, k-1) + \beta \sqrt{s_{ma}^2(n_{j_0}, j, k-1)}$$  \hspace{1cm} (2)

In Equation 2, $\beta$ is an adjustment parameter. In addition, when the latest PCV under the same environmental condition parameter of the CF is less than $\theta_2(k)$, the submodule performs the following procedure. Assuming a function corresponding to a larger value of the environmental condition parameter can handle more severe conditions with higher computational load, the submodule selects a function with a higher PCV of the environmental condition parameter than the current CF from the target functions other than the CF. In an example shown in Table 3, when the current environmental condition parameter is 5, the functions that can cope with severe environmental conditions are A, B, and C. To select a function other than the CF, function B is selected corresponding to the environmental condition parameter 20-29.

When the latest PCV is higher than $\theta_2(k)$, the CF is formally selected.

<table>
<thead>
<tr>
<th>Environment Condition Parameter</th>
<th>0-9</th>
<th>10-19</th>
<th>20-29</th>
<th>30-39</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantization slot No. $j$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>…</td>
</tr>
<tr>
<td>Function name corresponds to Function Identification Number</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>…</td>
</tr>
</tbody>
</table>

Operations and Observations Adapter Submodule

The O&O Adapter submodule executes the selected function based on the rules described in the previous subsection. The values of the sub-PCV and environmental condition parameters are obtained
from the results of the execution. The sub-PCV is a value which is a factor for determining the quality of service by the execution of a function. If there are multiple sub-PCVs, PCV \( x(n_{fid}) \) for the function corresponds to \( n_{fid} \) is:

\[
x(n_{fid}) = \sum_{i=1}^{N_{sb}} p_i w_i
\]

Equation 3 can be derived based on the multiple sub-PCVs, where \( N_{sb} \) is the total number of subcharacteristic values to consider, \( p_i \) is the \( i \)-th sub-PCV, and \( w_i \) is a weighting factor expressing the importance of the \( i \)-th sub-PCV. Examples of sub-PCVs are the response time to return a response result to the user \( p_1 \), cost of using the server \( p_2 \), precision of function output \( p_3 \), and the amount of information in the output of the function \( p_4 \).

**PCV Management Submodule**

The PCV Management submodule updates the moving average value, the moving variance value, and the latest PCV of the PCV map for each function. Let N be the total number of functions, Figure 6 describes the management of PCVs for each function as follows:

1. The horizontal axis of Figure 6 (i.e., environment condition parameter) is divided into a predetermined number \( N_{fs} \) of quantization slots.
2. Each quantization slot holds a moving average \( \mu_{ma} \) and a moving variance \( s_{ma}^2 \) at all times. Thus, even when there is a sudden outstanding value, the variation can be smoothed and the amount of data to be stored can be reduced.

**Figure 6. Example of data management method for Performance Characteristic Values**

In Figure 6, the large circle indicates the representative value, and the small circle indicates the past observation value. The data held are only a representative value (large circle) and a latest observation value (small circle) for each quantization slot.
Specifically, for each function identified by $n_{fid}$ and each quantization slot $j$, moving average of PCVs $\mu_{ma}(n_{fid}, j, k)$ can be derived using the average of the previous cycle $\mu_{ma}(n_{fid}, j, k - 1)$ and the latest PCV $x(n_{fid}, j, k)$:

$$\mu_{ma}(n_{fid}, j, k) = \frac{1}{w_s + 1} \left( w_s \mu(n_{fid}, j, k - 1) + x(n_{fid}, j, k) \right)$$

(4)

where $j$ is the quantization slot number (1, 2, 3, ..., $N_{fs}$), $k$ is the number of cycles (0, 1, 2, 3, ...), and $w_s$ is a weighting value to adjust how many past observation values are taken into consideration to derive the moving average and variance. As a result, it is possible to determine to what extent the past observation value or the latest PCV is more important.

For each function identification number $n_{fid}$ and each quantization slot $j$, the moving variance value of the PCVs $s_{ma}^2(n_{fid}, j, k)$ can be derived using the variance of the previous cycle $s_{ma}^2(n_{fid}, j, k - 1)$ and moving average $\mu_{ma}(n_{fid}, j, k)$:

$$s_{ma}^2(n_{fid}, j, k) = \frac{w_s}{w_s + 1} \left( s_{ma}^2(n_{fid}, j, k - 1) + \mu_{ma}^2(n_{fid}, j, k - 1) \right) + \frac{x^2(n_{fid}, j, k)}{w_s + 1} - \mu_{ma}^2(n_{fid}, j, k)$$

(5)

The PCV Management submodule updates and keeps moving average $\mu_{ma}(n_{fid}, j, k)$ and moving variance $s_{ma}^2(n_{fid}, j, k)$ for cycle $k$, for each function and quantization slot, based on the above calculation.

The function identification number $n_{fid,opt}(j)$ of the function having the highest moving average $\mu_{ma}$ at the cycle $k$ in the quantization slot $j$ with respect to the environmental condition parameter value is derived as follows:

$$n_{fid,opt}(j) = \arg \max_{n_{fid}} \mu_{ma}(n_{fid}, j, k)$$

(6)

As a result, the PCV Management submodule generates the decision table (see Table 2).

**Updating Data on the Server Side**

In order to accurately determine the adaptive operation in this method, it is important to keep and use the latest PCV of each function. However, regular execution of the function on the client side results in an unnecessary increase in calculation load. Since the execution of functions on the server side does not affect the calculation load on the client side, the system has a mechanism to reuse the result of other clients calling and executing functions on the server side and to update PCV of functions on the server side. In addition to utilizing the execution results of other clients, the authors built a periodic execution mechanism that automatically and periodically executes functions on the server side and updates the latest values.

The calculation time of the function of the server is independent of the client device, but the communication time with the server differs for each client, so the calculation time of the function...
on the server side is managed by the server, and the communication time between the server and the client is held and managed for each client. A client can estimate user response time by asking the server for the calculation time of the function, and adding up the communication time for the client.

**EVALUATION**

In this section, the proposed method is evaluated using the Buddyup! service by comparing it with the former version of the service which only performs nxm-matching in the client without using the proposed method.

**SYSTEM CONFIGURATION**

Figure 7 shows the system configuration adapted to Buddyup! from the method illustrated in Figure 3.

![Figure 7. Buddyup! system configuration](image)

The frontend deploys Buddyup! as Service Module (see SNS Service in Figure 7) and adds the Adaptation Module to enable adaptation. The backend uses Firebase, a cloud service, with Cloud Firestore as the database for storing information and Firebase Cloud Functions for executing server logic.

The adaptation operation of Buddyup! is configured to select an optimum matching function from the following three functions, in response to a matching request from the highlighting submodule:

1. Client mx1-matching function is deployed in the Service Module of the frontend and performs mx1-matching by using information in a client device. It is very lightweight and used as a fallback function if the response of the nxm-matching function is estimated slower than a certain threshold.
2. Client nxm-matching function is deployed in the Service Module of the frontend and performs nxm-matching by using information in a client device.
3. Server nxm-matching function is deployed in Firebase Cloud Function of the backend. It performs nxm-matching by acquiring matching information from the matching information database.

Table 4 compares the functions that can be performed by the existing and proposed methods. The existing method supports matching functions in the client by manually switching them prior to the event according to the number of participants. The nxm-matching was used only when the number of participants is expected to be small. The proposed method can switch between these three matching functions automatically.

Table 4. Matching functions in existing and proposed methods

<table>
<thead>
<tr>
<th></th>
<th>Existing method</th>
<th>Proposed method</th>
</tr>
</thead>
<tbody>
<tr>
<td>nx1-matching in client</td>
<td>ü (switch manually)</td>
<td>ü</td>
</tr>
<tr>
<td>nxm-matching in client</td>
<td>ü (switch manually)</td>
<td>ü</td>
</tr>
<tr>
<td>nxm-matching in server</td>
<td></td>
<td>ü</td>
</tr>
</tbody>
</table>

The Adaptation Module of Buddyup! uses the number of matched persons as the environment condition parameter, and the response time as PCV. The O&O Adapter submodule executes the matching function selected by the Control submodule, returns the result to the highlight submodule, and measures the response time. The measured response time is passed to the PCV Management submodule. The PCV Management submodule calculates the moving average for each quantization slot, and then updates the PCV database in the backend Cloud Firestore. The Periodic Execution function executes the server nxm-matching function using the Schedule function of Firebase Cloud Functions to reflect the latest status of the PCV map and the decision table to PCV database in Figure 7. In Buddyup!, the regular run of the Periodic Execution takes place every minute.

**EVALUATION SETTINGS**

The authors evaluated the actual operation and the effectiveness of for the system described in the previous subsection. In the evaluation of the effectiveness, the response times of the method used in this study has been compared with that of the existing method described in the previous section, using only nxm-matching in the client. The evaluation consists of the following three operations:

1. Switching Matching Functions.
2. PCV update during execution of the matching function.
3. Overhead of the adaptation process.

Figure 8 is a diagram and specifications for the evaluation environment, and Table 5 lists the parameters used in the evaluation.

For evaluation purpose, we needed to see if the switching of the matching function was done correctly by retrieving data from a wide range of people in a single event. However, this was difficult to fulfil in real events, where the number of people and the execution environment vary. Therefore, instead of collecting data from a real event, we simulated the event situation so that we could obtain data for a wide range of people. In the evaluation, a highlight function was executed from a client device by using a script, and the response times were recorded. In order to simulate the actual event,
participants’ information, which was prepared in advance, was added at one-minute intervals from another device using a script. The experiment mimicked each of the phases to simulate the actual event (see Table 6):

### Table 5. Parameters used in the evaluation

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w_1 ) Quantization slot width</td>
<td>2</td>
</tr>
<tr>
<td>( w_s )</td>
<td>0.538</td>
</tr>
<tr>
<td>( \beta * \sqrt{s_{na}^2 {n_{j, k} - n_{j, k - 1}}} ) [ms]</td>
<td>-5000</td>
</tr>
<tr>
<td>( \theta_1 ) [ms]</td>
<td>-120000</td>
</tr>
<tr>
<td>Periodic execution interval [ms]</td>
<td>60000</td>
</tr>
</tbody>
</table>

### Figure 8. Evaluation environment

![Evaluation environment diagram](image)

### Client Specifications

<table>
<thead>
<tr>
<th>Component</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Intel® CoreTM i7 -7500 U CPU @2.70 GHz</td>
</tr>
<tr>
<td>RAM</td>
<td>8.00 GB</td>
</tr>
<tr>
<td>OS</td>
<td>Windows 10 Pro 64bit version 1903</td>
</tr>
<tr>
<td>Browser</td>
<td>Google Chrome</td>
</tr>
<tr>
<td></td>
<td>Version: 80.0.3987.106 (Official Build)</td>
</tr>
</tbody>
</table>

### Server Specifications

<table>
<thead>
<tr>
<th>Component</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform</td>
<td>Google Cloud Functions</td>
</tr>
<tr>
<td>Processor</td>
<td>1.4 GHz</td>
</tr>
<tr>
<td>RAM</td>
<td>1GB</td>
</tr>
</tbody>
</table>
Phase 1: The initial state before the event starts.
Phase 2: Participants register for the event. The number of users and tags increases.
Phase 3: Participants complete their registration and edit their self-introduction text. Only the number of tags increases.

For the initial values of PCVs of each function, Figure 9 shows the measured calculation time (average of 10 executions) for each matching function.

**Table 6. Phases of the event**

<table>
<thead>
<tr>
<th>Phase</th>
<th>#participants</th>
<th>#tags</th>
<th>#matched persons</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Before start</td>
<td>5</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>Immediately after the start</td>
<td>From 5 to 300</td>
<td>From 25 to 900</td>
</tr>
<tr>
<td>3</td>
<td>In the event</td>
<td>300</td>
<td>From 900 to 1500</td>
</tr>
</tbody>
</table>

**Figure 9. Initial value of PCVs**
EVALUATION RESULTS

Effectiveness of the Method

Figure 10 depicts the change in response time between the existing method and the method used in this study.

In the method used, when the number of matched persons was less than 29, the matching was done in the client in the same manner as in the existing method. When the number of matched persons was 29 or more, the proposed method showed faster result than the existing method. This was because the Adaptation Module switched the operation to the matching function in the server and improved the response time. Previously, Buddyup! had to be configured to run only nx1-matching when the number of participants was expected to be large, which significantly reduced the CX. The proposed method maintained the response time of the nxm-matching function within 6 seconds even if the number of matched persons reached 50, and enabled to provide the nxm-matching function without impairing the CX even if the event became large.

EVALUATION OF THE BEHAVIOR

Switching Matching Functions

Figure 9 shows the PCVs used for switching the matching function of the server and the client were flipped at the number of matched persons of 29. At this point the server and client execution functions were switched dynamically.

Table 7 explains this more specifically. Of the three attempts with 29 matched people, the first attempt was performed in the server because the server had a smaller (i.e., higher performance) value of the PCV registered in the initial value. The actual response time was 6315 ms, and the PCV of the server was updated based on this. On the second attempt, the value was smaller for the client and
the client matching function was chosen. Here, the PCV was updated to 2631 ms according to the response time of the client, and the client function was executed in the third trial.

As a result, the switching of the matching function reflected the result of the function execution and returned the faster response. It means the mechanism worked correctly to maintain the best response time.

**Updating PCVs by Matching Execution**

In order to check that each PCV was updated after each matching function execution, the initial PCV was compared with the updated result after the experiment. Figure 11 highlights the change of PCVs after the completion of all trials. PCVs of both the client and the server were updated after the experiment.

**Table 7. Results of 29 matched persons**

<table>
<thead>
<tr>
<th>#try</th>
<th>Matched #persons</th>
<th>PCV client [ms]</th>
<th>PCV server [ms]</th>
<th>Function</th>
<th>Response Time [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>29</td>
<td>5000</td>
<td>2259</td>
<td>Server</td>
<td>6315</td>
</tr>
<tr>
<td>2</td>
<td>29</td>
<td>5000</td>
<td>6315</td>
<td>Client</td>
<td>2631</td>
</tr>
<tr>
<td>3</td>
<td>29</td>
<td>2631</td>
<td>6315</td>
<td>Client</td>
<td>2016</td>
</tr>
</tbody>
</table>

As a result, the switching of the matching function reflected the result of the function execution and returned the faster response. It means the mechanism worked correctly to maintain the best response time.
The PCV of the server is expressed as the sum of sub-PCVs, which are communication time and function execution time. The initial values of PCVs are registered in advance. On the client, the value of PCV does not change significantly before and after the execution, suggesting that there is not much other load or environmental factors to the client device. On the other hand, for the server, while the PCVs are almost constant before the execution, they change after the execution.

As shown in Figure 10, when the number of matched persons is less than 29, the client function has been selected, hence the results of the periodic execution are reflected here. This experiment is repeated for each number of matched persons, i.e., from 1 to 50. The server logic is selected from 27 persons and the graph changes from here.

There are two possible reasons why the numbers are often larger than the small number of matched persons. One is the load on the network. As the periodic execution doesn’t incur the network communication, the results are calculated based on the result of the execution and a constant network delay measured before the experiment. The other is that the server is running in a multi-tenant environment, and there was an influence from other loads.

**Overhead of the Adaptation Process**

To identify the overhead of the adaptation process itself, the execution time has measured by comparing execution time of the modules with and without the adaptation mechanism.

As a result, the average processing time in the Control submodule is 6.0 msec ($\sigma = 4.1$ msec), and the average processing time in the O&O Adapter submodule and the PCV management submodule is 4.8 msec ($\sigma = 3.3$ msec), which is sufficiently small compared to the time required for the matching process. Thus, the proposed mechanism will not impact the overall process.

**RELATED WORKS**

As outlined in Target Application subsection, this research focuses on web applications that support event-based personnel exchanges. In these applications, the participant needs to repeatedly contact the other participants within a limited time frame and extract data from them to find the desired candidate. Therefore, the matching result needs to be provided immediately upon the user’s request, and both the matching result and response time have a substantial impact on the user experience. As the response time is affected by the input at that time, the system doesn’t have dead time, time constants, time delay or hysteresis. There is no need to consider the stability of the control. The response time is highly dependent on the process state of the operating system in the client device, the network congestion, the change in the number of participants, and the change in the number of tags held by each user.

The position of the proposed method is described in Table 8. Methods that take into account temporal dependence and adaptive behavior with state transition models include prediction and optimization using stochastic models (Cailliau & Van Lamsweerde, 2017) and adaptive decision making by preparing a model of the Markov decision process off-line in advance and using different models depending on the situation (Moreno et al., 2018). There are methods using model predictive control (Angelopoulos, Papadopoulos, Souza, & Mylopoulos, 2018) and SimCA* as an innovative self-adaptive framework using PID controllers and multi-objective optimization (Shevtsov et al., 2019). Depending on the size of the prediction model and the number of possible combinations of alternatives that can be taken, a challenge for these approaches is that the computation time for prediction, optimization and control could be long. Methods based on reinforcement learning (Torabi, Wenkstern, & Saylor, 2018) have the ability to flexibly respond to such environmental changes, but they need long time for learning.

In response to this, Cyber Foraging (Mahadev, 2001) allows a powerful server to take over the computationally demanding processing. Moghaddam, Procacciantia, Lewis, & Lago (2018) evaluated Cyber Foraging strategies from the viewpoints of fault tolerance and energy cost, Akbar, & Lewis
(2018) evaluated Cyber Foraging strategies based on trade-offs between energy consumption and computation time using representative models of self-adaptive and self-aware when performing Cyber Foraging in robots, and Ventrella, Esposito, & Grieco (2018) proposed FORMICA as an architecture for Cyber Foraging that minimizes the completion time of a series of jobs offloaded from mobile devices. A problem is when environmental changes occur, such as the number of participants in an event or the network status of the venue, the prediction model needs to be tuned again.

Our method uses a PCV map with update rules, threshold rules, and a learning function to immediately determine which functions should be executed, and the learning function allows the method to be used under changing environment without tuning the prediction model. The proposed method also features a simplified function as a fallback function to reduce computation time. The PCV map can learn incrementally as the environment changes, and the threshold decision is robust to rapid environmental changes. The server-side can learn the PCV map from other users’ execution as well as periodic execution.

**Table 8. Comparison with other researchers**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptation target</td>
<td>Without time constants / dead time</td>
<td>With time constants / dead time</td>
<td>With time constants / dead time</td>
<td>With time constants / dead time</td>
<td>Without time constants / dead time</td>
</tr>
<tr>
<td>Adaptation means</td>
<td>Data-driven (learning map)</td>
<td>Model-based optimization</td>
<td>Model-based control</td>
<td>Model-based control</td>
<td>Model-based optimization</td>
</tr>
<tr>
<td>Cyber Foraging(run on server)</td>
<td>Not stated</td>
<td>Not stated</td>
<td>Not stated</td>
<td>Not stated</td>
<td>Not stated</td>
</tr>
<tr>
<td>Fallback function</td>
<td>Not stated</td>
<td>Not stated</td>
<td>Not stated</td>
<td>Not stated</td>
<td>Not stated</td>
</tr>
<tr>
<td>Decision in short time</td>
<td>Depends</td>
<td>Depends</td>
<td>Depends</td>
<td>Depends</td>
<td>Depends</td>
</tr>
<tr>
<td>Robustness for sudden change</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>×</td>
</tr>
<tr>
<td>No retune required</td>
<td>(learning)</td>
<td>×</td>
<td>×</td>
<td></td>
<td>×</td>
</tr>
</tbody>
</table>

**CONCLUSION**

In this paper, the authors proposed a mechanism to manage the performance history and dynamically switch execution positions and contents of functions to improve CX in mobile applications. Each function has a PCV map, and an appropriate function and its execution position are selected according to environmental conditions. The PCVs and the sub-PCVs constituting them may be calculated for each user or may be one overall. By combining sub-PCVs, the application can estimate the expected user experience such as a response time for a particular function of a particular user. Depending on the calculated PCV, the CX is maintained by selecting a fallback function. As the PCV map is updated
regularly, the executing function can be appropriately selected in a cloud environment where load and the number of users continue to change.

The authors implemented the mechanism in the matching function of a cloud application for personnel exchange events called Buddyup! and confirmed its effectiveness. The method significantly shortened the response time even in computationally intensive nxm-matching function, and allowed the application to provide matching results within 6 seconds, which was not available in this application without the proposed mechanism. As a result, it could provide matching without sacrificing CX even in a large-scale event with 300 people.

In Buddyup!, as the server is using Firebase Cloud Functions for the server and the implementation language is Node.js (JavaScript), porting the matching function from the client to the server with little extra implementation effort.

The future work includes more case studies in real events and the use of feedback from users. By obtaining feedback from users, the functions can be prioritized based on the preference by users. In such a case, both the execution performance and the preference based on the user feedback become sub-PCVs, and the CX can be further improved by dynamically changing the weighting of the sub-PCVs.

**FUNDING AGENCY**

Publisher has waived the Open Access publishing fee.
REFERENCES


Taka Matsutsuka received his M.S. degree in Computer Science from Tokyo Institute of Technology, Tokyo, Japan in 1996, and received his PhD from Akita Prefectural University, Japan in 2021. He was a visiting researcher at Carnegie Mellon University in 2001-2002, and was engaged in research of pervasive computing. He was working for Fujitsu Laboratories, and engaged in a wide range of software-related research activities including distributed systems, enterprise architecture, data engineering, and self-adaptive software until 2021. He is working for Fujitsu Limited and engaged in data exchange business. He is a visiting professor at Japan Advanced Institute of Science and Technology (JAIST). He is a member of Information Processing Society of Japan (IPSJ). He received his M.E. and Dr. Eng. degree from Waseda University, Japan, in 2005 and 2008, respectively. He worked for Waseda University from 2007 to 2011. Since 2011, he has worked for FUJITSU LABORATORIES LTD. He received the outstanding paper award by general chair of ICCAS2013 from the Institute of Control, Robotics, and Systems (ICROS) in 2013, the best paper award from the Society of Instrument and Control Engineers (SICE) in 2013, the young author award from SICE in 2015 and the technology award of conference on control division from SICE control division in 2017. His current research interests are machine learning, modeling, simulation, optimization and control for an industrial system. He is a member of SICE and the Japan Society of Automotive Engineers (JSAE).

Yohei Toriyama received his master’s degree in Precision Engineering from the University of Tokyo for his research on image processing for imaging result of atomic force microscopy. He was engaged in research and development of computer-aided construction at Hitachi Construction Machinery from 2014 to 2019. He joined Fujitsu Laboratories Ltd. in 2020, and was engaged in research and development on self-adaptive systems and asynchronous data processing platform. Since Fujitsu Laboratories Ltd. was absorbed and merged by Fujitsu Limited in 2021, he has been engaged in research and development on trust service for data exchanging between cloud storage services in Fujitsu Limited.

Noriyasu Aso received a master’s degree in manufacturing science from Osaka University. Since joining Fujitsu Laboratories in 1999, he has been engaged in R & D on photocatalysts for mobile electronic devices and energy conservation technology using waste heat from data centers, engine combustion control using artificial intelligence at Transtron Inc. from 2015 to 2018, self-adaptive systems and design automation technology for industrial systems at Fujitsu Laboratories from 2019 to 2020. Since 2021, he has been engaged in research and development on social model integration at Fujitsu Limited. His research interests include thermodynamics, machine learning, control technology, modeling, optimization, and artificial intelligence. Ichiro Iida was born in Kyoto, Japan in 1955. He received the B.E., M.E. and PhD degrees from the University of Tokyo in 1978, 1980, and 1983, respectively. Since joining Fujitsu Laboratories Ltd. in 1983, he has been engaged in research and development of Information networking systems and mobile computing. In 2016 he moved to Akita prefectural University as a professor of the department of Information and computer science, where he applied the mobile technologies to regional issues such as smart agriculture and sharing economy. His research interests include Internet architecture, Internet of Things and Cyber Physical Systems. He is currently one of the board members of the university, responsible for research management and regional contributions.