Customized Disease-Driven Request Management in Uplink, Update and Synchronize System of e-Healthcare

U-Hou Choi, Jia-Li Ma, and Ming-Chui Dong
Dept. of ECE, FST, University of Macau, Macau S.A.R., China
Email: [mb15509, mcdong]@umac.mo, jimohanqiu@hotmail.com

Abstract—The embedded device based distributed e-home healthcare system brings lots of convenience for prevention and detection of Cardiovascular Diseases (CVDs) by means of on-site multi-vital-signs acquisition and local intelligent point-of-care diagnosis functionalities. Due to the intrinsic source restriction, a backyard remote Uplink, Update and Synchronize (UUS) system is imperative to manage the uplinked help requests from mobile devices and feedback promptly the update package. Processing thousands of concurrent requests is of most significant in the request handling system design. The existing request scheduling schemes are various but each has its own pros and cons. Through the further analysis of UUS and the study on different scheduling schemes, a novel dynamic request management system is proposed. The pioneer equation adapted in this system quantizes representative factors for request processing sequence evaluation which makes handling large quantity of concurrent requests in an effective and efficient manner possible.

Index Terms—cardiovascular diseases diagnosis, distributed e-home healthcare, server-client system, request management, request scheduling, dynamic priority.

I. INTRODUCTION

Nowadays distributed e-healthcare systems combining intelligent mobile devices, wireless data exchange, and remote diagnosis bring lots of benefits to conventional medical system by transforming the rescue-centered healthcare practice to patient-prevention-centered approach. However, limited networking resources and insufficient Knowledge Base (KB) on clients’ side impedes its development on covering variety of diseases. Therefore, a powerful reinforcement system is definitely a cornerstone for building an intelligent distributed e-healthcare system.

In the previous work [1] and [2], an intelligent distributed e-home healthcare platform was proposed to resolve the rapidly increasing demands on e-healthcare services and reduce the burden on hospitals and medical personnel. As illustrated in Fig. 1, the three sub-systems, representing different roles in this architecture, are linked-up by a backend assistance system named Uplink, Update and Synchronize (UUS) system. The UUS manages the data exchange activities and schedules process tasks in these sub-systems automatically. The mid- / downstream systems not only act as signal acquisition terminals, but also support the local diagnosis based on the Resided Knowledge Base (RKB) pre-installed. With the coordination of UUS, clients could request help from the remotely-linked upstream only when support is required, therefore can highly reduce dependence on servers, in terms of unnecessary traffic and server workload.

Figure 1. Systematic architecture of distributed e-home healthcare platform.

The concept of UUS is basically a server-client or web-cloud mechanism, in which mid-/downstream systems initiate requests to upstream as client and are waiting for response. The upstream acts as server, processes received requests and responses correspondingly and promptly. For instance, when a client encounters diagnosis failure due to insufficient knowledge in his/her RKB, it prepares a request with relevant information assembled, and sends it to upstream for analysis under client’s permission. The server in upstream is equipped with abundant KB rules and comprehensive Data Base (DB) storing all clients’ information, thus is able to recognize and reply clients individually the accurate, detailed diagnostic results as well as select absent knowledge for client’s RKB update.

The UUS is not only a helpdesk system or ordinary client web portal for issuing requests to servers, it is a customized disease-driven request management system designed for controlling the request-response as well as customizing RKB automatically and intelligently in distributed e-healthcare architecture. The major
challenges come from the fact that each client’s request actually involves RKB rule update and/or upstream diagnosis with personalized data, thus data broadcast, which is a common way to satisfy multiple user requests simultaneously, is not applicable. Furthermore, some particular requests may have higher severity and urgency, e.g. if a client’s recent diagnostic results show continues deterioration of health status and reach risky level, the client may probably request upstream diagnosis to confirm urgently. Besides, the server must be able to process and response requests in time because large number, various types of user requests arrive at the upstream server continuously. Otherwise, as the number of pending requests grows, the server might be overloaded eventually and cause system unavailable.

Tackling these aspects, an intelligent request scheduling algorithm is a must to deal with large quantity, diverse requests. In [3], several conventional request scheduling algorithms such as First-Come-First-Served (FCFS), Shortest Request First (SRF), and etc. were studied but most of them are single factor oriented, designed for common data broadcast, or lack of capability in adjusting the process sequence based on current situation. In [3]-[6], the improved priority based schedulers were proposed but each one has its own features tackling specific data structure or application, which still could not fully conform to UUS’s requirement. Hence, a novel customized disease-driven request scheduling algorithm is proposed in this paper, which has ability to quantize multiple factors involved in the scheduling in a simple but effective way, as well as to update dynamically the overall sequence of pending requests based on real-time resource status.

II. UPLINK, UPDATE AND SYNCHRONIZE SYSTEM ARCHITECTURE

As depicted in Fig. 2, the UUS system is constructed by five functional modules which links down-, mid-, and upstream together. The first request generation module initializes the requests from client devices and delivers request information through the uplink. The request could be triggered manually by client, e.g. user can choose asking to download additional knowledge of specific diseases for customizing RKB, and also the system would provide options for user to trigger more accurate and comprehensive upstream diagnosis especially when severe disease is detected during local diagnosis. The request could also be triggered automatically by the system when diagnosis failure is occurred due to insufficient knowledge in RKB. Once the request is initialized, system can automatically determine the most suitable request type, collect and package the relevant information, including request type code, user personal information, acquired multi-vital signs, and local diagnostic results, into one information file and deliver it to the upstream.

The second part is the request pretreatment taking place on the upstream side. The request scheduling algorithm is implemented in this module. Upon receiving the request and information file, the data will be stored and analyzed so that the request processing sequence is determined. The confirmed sequence, in a form of pending request list, will be loaded into the third part—the customized disease-driven request management module, which is responsible for request analysis including the upstream diagnosis and necessary judgment of updating client’s RKB. This module records the used rules during the upstream diagnosis, and if RKB update is needed, the absent rules in client’s RKB will be filtered out and packaged into an update file before delivering the upstream diagnostic results to client.

When the upstream response has successfully delivered to the lower stream, the accurate upstream diagnostic results will be displayed to user. In case RKB update file exists, the RKB update module will update the client’s RKB automatically. Detailed design of RKB update scheme has been proposed in previous work [2]. Upon update success, the lower stream will feedback a confirmation to upstream for DB synchronizing so that it grasps always the up-to-date RKB condition on each client’s device in terms of what knowledge rules exist. This functionality is in charged by the synchronize module on server. Until this point, the request handling cycle is completed and the corresponding request will be removed from the pending request sequence list.

III. HELP REQUEST SCHEDULER

A. Technical Requirements

As mentioned, the UUS’s core feature is actually an intelligent disease-driven request scheduler managing the uplink help requests. Referring to the reviewed bottleneck problems that UUS may encountered, the most significant...
competences of the request scheduler can be summarized as followings:

1) Customization – Each help request and its involved user data are individual, thus customization processing is necessary for every individual request;
2) Prioritization – Help requests have different disease severities which are related with different emergencies and require consequently the priority treatment;
3) Efficiency – Large number of help requests might be received by upstream concurrently, thus timely response to each individual request must be ensured to avoid system congestion and eventually timeout which requires optimizing processing resource allocation;
4) Simplicity – A simple algorithm design may minimize resource consumption as well as increase compatibility on various platforms.

In [2], an intelligent RKB management system with convenient and effective RKB design is proposed, which can highly reduce the system resources in data processing. Based on estimation and testing, the size of plain text KB file containing 100 rules is within 10 kilo-byte, and the diagnostic program can complete prognosis within 0.3 seconds, while the client’s RKB update process takes around 0.1 seconds on average. Even including the seconds, while the client’s RKB update process takes

the statistics from practical operation. A well-designed request scheduler can help optimizing resource utilization and maximize operation efficiency. The conventional request schedulers have been reviewed and improved for various applications [3]-[6], feasibility of applying them in UUS is analyzed as followings:

- FCFS is a conventional scheduling approach which processes the requests according to the arrival sequence, thus it is very easy to implement. However, it lacks of intelligence to consider each request’s characteristics.
- SRF handles the easiest request first, i.e. request with the shortest estimated processing time and thus the response speed and success rates could be highly boosted. However, the important but complicated requests could be slighted, which is not preferred in UUS.
- Earliest Deadline First (EDF) schedules data items based on their deadlines, i.e. the priority depends on the request’s severity. EDF is a classical scheduling algorithm in real-time systems, which shows good response rate. But similar to FCFS and SRF, it concerns only one characteristic of request, which is not comprehensive enough for UUS.
- Most Request First (MRF) firstly broadcasts the most popular data item in the pending requests. However, in UUS requests are independent, namely no exactly similar data item is required.
- Longest Wait First (LWF) replies the data with the largest total waiting time from all pending requests. As no common data in UUS requests, this algorithm only considers each request’s net waiting time, which is similar to FCFS.
- Number of pending Request multiply Waiting time (RXW) broadcasts the data item with maximum value of R (total number of pending requests for a common data item) multiplying by W (the amount of time that the earliest arrived pending request has been waiting). This approach combines benefits of FCFS and MRF, thus provides balance between waiting time and data popularity, but this is not applicable for UUS due to no common data exists.
- Slack time Inverse Number of pending requests (SIN) broadcasts the data item with the minimum SIN value, which is defined as the ratio of slack time to the number of pending requests for the item. SIN integrates both data popularity and request urgency, but similar to RXW and other data tackling common data access, it is not favorable for UUS.
- Static Priority (SP) scheduling algorithm generally refers to those algorithms which calculate the priority of each arrived requests before start scheduling process. Usually the priority calculation is multi-rules based. Comparing to the aforementioned straight-forward algorithms, SP is more comprehensive by considering various factors, but the priority is fixed in scheduling process and cannot be adjusted or adapted anymore.
- Dynamic priority (DP) scheduling algorithm inherits the advantages of SP with additional adjustable queuing by re-calculating the request priority in each cycle. This is more flexible in multi-factors/criteria based calculation and gives good balance in resource consumptions, request severity and waiting/retrial times of pending requests.

Most of the reviewed algorithms are not applicable in UUS since they only consider single scheduling criteria, such as arrival sequence or waiting time, or are mainly designed for common data broadband. SP and DP have good flexibility in defining customized scheduling
criteria, and become popular in pace with powerful computing capability and multi-tasking competence of modern servers. Tackling the technical requirements of UUS, a novel priority-based disease-driven request scheduler is proposed. The system is deployed as a whole on the five modules as depicted in Fig. 2, therefore each part’s functionality is described separately.

C. Novel Disease-Driven Request Scheduler

1) Disease-driven request generation

The knowledge management system on mid-/downstream embedded device requests user registration with a unique user name (user ID) and his/her basic personal information, e.g., age, gender, height and weight as part of inputs for diagnosis. With real-time measured multi-vital-signs, the local diagnosis is performed on user client device based on client’s RKB. The diagnostic results are then displayed to user with option to trigger upstream help request, which is especially helpful when diagnosis failure encountered, or when severe disease or symptom implying potential severe disease is being detected. Noted that the client’s pre-installed RKB has only the basic knowledge at the beginning, users can customize his/her RKB by downloading RKB update package proactively, thus not necessary to update until occurring diagnosis failure.

Each request requires customized processing and consumes different system resources. In order to deal with huge amount of concurrent help requests, the system resources must be wisely allocated to avoid response timeout or even system congestion. Therefore, the request type and corresponding procedures (workload) on upstream should be standardized in the initial design.

According to the condition the client encountered, there are several types of request as listed in Table I.

To prepare a request, the client system packs all essential information into a file containing the fore-mentioned user ID, personal data, measured medical signals, request type (type number only), and diagnostic results only if no diagnosis failure encountered. The client system always initiates the communication to upstream. After the link establishment acknowledgement is received, client system delivers the help request file to upstream for further processing.

2) Disease-driven request pre-treatment

The arrived requests are stored in upstream system and distinguished by the unique user ID attached with. Based on pre-defined rules, a help request’s processing sequence is determined in this module before entering the actual handling process. The concept is similar to the triage station in hospital categorizing and prioritizing waiting patients. Multiple factors, either provided by client or server, should be involved for comprehensive decision making. However an intricate calculation may increase system complexity, consumed resources which is actually the cost, eventually degrade the efficiency. Considering the system simplicity, only the following most representative factors are chosen and quantized for request processing sequence evaluation:

- User Group
- Request Type
- Retry Times

As shown in Fig. 3, these factors have different levels and corresponding ratings in evaluation. In the design, higher final score should be in the top of sequence list, i.e. to be handled in first place. The factor “user group” is determined in upstream after analyzing the user data packed in help request and compared with previous diagnostic records storing in upstream DB. Users are categorized into normal group or critical group (users in

---

### Table I. User Client Request Type Classification

<table>
<thead>
<tr>
<th>Type Number</th>
<th>Request Type</th>
<th>Detail Description</th>
<th>Process on Upstream</th>
<th>Service Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>RKB Update Request (Optional)</td>
<td>User manually triggers RKB update request for client’s RKB customization.</td>
<td>Prepare knowledge rule package for client’s RKB update</td>
<td>L1</td>
</tr>
<tr>
<td>1</td>
<td>Diagnostic Results Confirmation (Optional)</td>
<td>Diagnostic results show no any severe disease or no any critical symptom of severe disease being detected; Optional for user to trigger manually.</td>
<td>Run upstream diagnosis</td>
<td>L1</td>
</tr>
<tr>
<td>2</td>
<td>Diagnostic Results Confirmation (Optional)</td>
<td>Diagnostic results show critical symptoms of severe diseases being detected. Optional for user to trigger manually the upstream diagnosis for confirmation.</td>
<td>Run upstream diagnosis</td>
<td>L2</td>
</tr>
<tr>
<td>3</td>
<td>Diagnostic Results Confirmation for High Risk Diseases</td>
<td>Diagnostic results show severe disease being confirmed; System triggers the upstream diagnosis automatically for confirmation and starts early treatment. Additional knowledge rules of those particular diseases could be updated to client’s RKB.</td>
<td>1. Run upstream diagnosis  2. Prepare additional rules of that particular diseases for client’s RKB update</td>
<td>L3</td>
</tr>
<tr>
<td>4</td>
<td>Upstream Support for Diagnosis Failure</td>
<td>Diagnostic failure due to lack of knowledge and suitable rules in client’s RKB; System triggers upstream diagnosis and RKB update request automatically.</td>
<td>1. Run upstream diagnosis  2. Prepare absent rules for client’s RKB update</td>
<td>L3</td>
</tr>
</tbody>
</table>
bad health condition or have critical diseases detected). The second factor “request type” has already been introduced in request generation section. In table I, the “Service Level” column indicates the preferred process priority of that type of request whereas L1 and L3 represent the lowest and highest level, respectively. Requests triggered automatically due to diagnosis failure or high risk diseases detected should have highest priority. However they consume more upstream resources because the process includes upstream diagnosis and update file preparation. Fortunately, upstream diagnosis requests are not expected to occur frequently thus will not be a significant burden for upstream server.

In DP scheduling algorithm, one improvement from SP is that the request processing sequence is further adjustable after first evaluation. This advantage can be applied in UUS to avoid continually-incoming prioritized requests engross all system resources, making other waiting requests timeout. The factor “retry times” contributes more in the rating according to the numbers of failed attempts from a particular user. The number of attempts from particular users is counted in the pre-treatment module. Note that help request’s sequence from particular user is not updated periodically or whenever new requests come. Instead, the request processing sequence could be updated after a new evaluation for the automatic re-attempt from the similar client system after previous request timeout. This approach could highly reduce the processing workload on upstream since the evaluation takes place only when new request from the similar client system arrives, and not to re-evaluate the ratings for all pending requests.

\[ \text{User Group: Value = 1 for critical users; value = 0 for normal users} \]

\[ \text{Req_Type: Refer to table I, value = 0 for L1; value = 1 for L2; value = 2 for L3} \]

\[ \text{Retry_Times: Value = 0 for 1st attempt; value = 1 for < Nth retry; value = 2 for > Nth retry} \]

The number \( N \) is a configurable parameter which indicates the maximum retry times, i.e. after \( N \) times of continuous attempt, \( retry\_times \) will reach its maximum value and will no longer increase. This is a preventive measure for unexpected situation encountered in downlink mid-/downstream devices.

The parameter setting of weighting factors could highly affect the performance of disease-driven request scheduler, whereas the server performance, the actual user size and their behaviors (e.g. the request peak hour of a day, such as after meal or at night time) will affect the real time performance as well. To simplify the initial parameter analysis, it is assumed that the server has capability to handle the designed maximum concurrent requests where its performance maintains. This designed value is fixed in the analysis, but should be adjusted during practical run. Firstly, without any impact of weight factor, i.e. \( W_s = W_r = W_t = 1 \), the sum of the three factors ranges from 0 to 5, in which the proportion of \( user\_group : req\_type : retry\_times = 1 : 2 : 2 \). The weighting factor should be designed in integer and the total rating will be 100. In this case, the rating is straight forward to be compared in the design whereas avoiding decimal number calculation is beneficial because it could improve the program processing speed.

In the initial setting, the impact from the three selected factors should be balanced. As their original proportion is \( 1 : 2 : 2 \), then the proportion of weighting factors should be \( 2 : 1 : 1 \) in this case. However, due to the fact that \( user\_group \) can only be 0 and 1, then a large value of \( W_s \) will cause a great step forward in the final \( Req\_Rating \) for critical users. The weighting between three factors should be carefully designed to avoid monopoly from critical users – an extreme case that L1 requests from critical users are always being handled faster than L3 or L2 from normal users under equal conditions. During normal operations, as system resources are sufficient for all incoming requests in theory, \( req\_type \) basically indicates the workload condition on the server. Therefore \( W_s \) could be adjusted to hold a more significant part in \( Req\_Rating \). Besides \( W_s \), \( retry\_times \) will be affected by the setting of \( N \). The value of \( N \) could be set larger during peak hour since there will be many concurrent requests and suppose retrials are common. Based on the above concerns, Table II lists out the recommended parameter settings during various scenarios.

With this concurrent requests scheduling and request priority rating equation, the upstream system resources could be utilized in a much more flexible way. Even limited-source servers could handle concurrent requests wisely without significantly affecting its performance. The adjustable weighting factors can help optimizing the system during actual operation, e.g. adapting various user
size or user practices and fine-tune along with the system and user growth.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>$W_x / W_y / W_z$</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>40 / 15 / 15</td>
<td>The three factors contribute similar impact to the final $\text{Req_Rating}; N$ is 5 by default; Fine-tune is necessary as critical users monopolize almost all the upstream resources.</td>
</tr>
<tr>
<td>Normal Time</td>
<td>20 / 30 / 10</td>
<td>The $\text{req_type}$ plays an important role in priority evaluation; Low priority request type from critical users will be processed more legitimately; $N$ could be a small value, e.g. 2 to 4, when system resources are sufficient.</td>
</tr>
<tr>
<td>Peak Hours</td>
<td>20 / 20 / 20</td>
<td>The proportion of $\text{retry_times}$ in the $\text{Req_Rating}$ is slightly increased due to some retry attempts may happen during peak hour. The value of $N$ could be set higher, e.g. 6 to 10, in order to have larger buffer for retrying requests but not boosting the rating too fast.</td>
</tr>
</tbody>
</table>

3) Customized disease-driven request handling

According to the request sequence list from pre-treatment module, this customized disease-driven request handling module searches and reads the previously stored request on upstream DB according to user ID. The actual process depends on the request type as listed in table I. The upstream diagnosis utilizes the abundant KB equipped on upstream to analyze the received user personal data and multi-vital signs. The diagnostic result is highly accurate and comprehensive. The result message will be packaged with relevant information, and feedback to mid- / downstream, so that the results could be visible on client’s mobile devices. In case that the upstream diagnosis is triggered due to local diagnosis failure, the upstream should perform a mid- / downstream RKB absent rule analysis, and prepare a customized update file for client’s RKB update. Since client registration, the client’s RKB version is stored on upstream and synchronized between servers and clients during each update request [2], the upstream should be able to complete the absent rule analysis without any additional information from mid- / downstream. Based on the upstream diagnostic result, the client might be concluded as having certain diseases, and thus the system identifies which particular knowledge (rule package for those concluded diseases) is unavailable in mid-/ downstream RKB. The absent rules are then quickly filtered out and written into a RKB update file. The upstream diagnostic results and RKB update file will then be delivered back to client.

4) Diagnostic results and knowledge base update

Upon receiving the upstream diagnostic result file, the client system of down- and midstream immediately display the comprehensive and accurate results to users. Meanwhile, update the client’s RKB.

5) Synchronize

After successfully update client’s RKB, the client feedbacks a confirmation message to upstream, so that the upstream could keep track on this particular user’s RKB version. Moreover, the confirmation message also indicates that the request has been completed and the occupied system resources could be released for handling the next request on the sequence list.

IV. CONCLUSION

In this paper, the technical requirements and challenges of UUS have been analyzed and identified. UUS is able to offer customized disease-driven request management, in which properly carry priority treatment for severe patients or emergent requests, and efficiently handle large number of concurrent requests. The proposed request scheduling algorithm decides the request processing sequence automatically. Such a UUS is promising to be used on mobile devices through server-client or web-cloud mode.

ACKNOWLEDGMENT

This work is supported by the Research Committee of University of Macau under Grant No. MYRG2014-00060-FST and Science and Technology Development Fund (FDCT) of Macau Government under Grant No. 016/2012/A1

REFERENCES


U-Hou Choi received his B.Sc. degree in Electrical and Electronics Engineering from University of Macau in 2009. He is currently a master student in EEE at University of Macau. His main research interests are system automation, AI and its application in biomedical engineering.

© 2015 Journal of Life Sciences and Technologies
Jia-Li Ma received the M.Sc. degree in Electrical and Information Engineering from Tianjin University, Tianjin, China in 2015. Currently, she is pursuing master degree in Electrical and Computer Engineering at University of Macau. Her research interests include biomedical signals compression and construction of embedded-link e-health systems.

Ming-Chui Dong received M.Sc. degree in EEE in 1975 at Tsinghua University, China, Visiting Scholar in EEE in 1981 at Rome University, Italy. He is a Full Professor and PhD supervisor of ECE Department of University of Macau, Automation Department of Tsinghua University, China and Professor of YanTai University, China. His main research interests are AI and its application in biomedical engineering, CIMS, fault diagnosis, AI in vision-to-text, voice-to-text machine translation etc.