Journal of Korea Multimedia Society Vol. 20, No. 5, May 2017(pp. 808-826)
https://doi.org/10.9717/kmms.2017.20.5.808

Design and Implementation of MEARN Stack-based Real-time Digital Signage System

Trinh Duy Khue[†], Thanh Binh Nguyen[††], UkJin Jang[†],
Chanbin Kim[†], Sun-Tae Chung[†††]

ABSTRACT

Most of conventional DSS's(Digital Signage Systems) have been built based on LAMP framework. Recent researches have shown that MEAN or MERN stack framework is simpler, more flexible, faster and more suitable for web-based application than LAMP stack framework. In this paper, we propose a design and implementation of MEARN (ME(A+R)N) stack-based real-time digital signage system, MR-DSS, which supports handing real-time tasks like urgent/instant messaging, system status monitoring and so on, efficiently in addition to conventional digital signage CMS service tasks. MR-DSCMS, CMS of MR-DSS, is designed to provide most of its normal services by REST APIs and real-time services like urgent/instant messaging by Socket.IO base under MEARN stack environment. In addition to architecture description of components composing MR-DSS, design and implementation issues are clarified in more detail. Through experimental testing, it is shown that 1) MR-DSS works functionally well, 2) the networking load performance of MR-DSCMS's REST APIs is better compared to a well-known open source Xibo CMS, and 3) real-time messaging via Socket.IO works much faster than REST APIs.

Key words: Digital Signage System, Content Management System, Real-time Web Service, MEAN Stack, MERN Stack, REST API, Socket.IO

1. INTRODUCTION

According to the proven effectiveness of digital signboards for advertisement and public information, there has been a significant increase in the demand for digital signage systems (DSS) nowadays[1]. A digital signage system usually consists of content management system(DSCMS) and signage content players (responsible for displaying signage contents onto display devices).

In addition to primary task of contents display on schedule, DSCMS needs to support other tasks; monitoring of signage players about their status and what they are displaying, audience measurement, disaster information distribution, instant messaging, and so on [2]. Many tasks such as monitoring and disaster information distribution requires to be processed in real-time. The front-end GUI to back-end DSCMS server, called dashboard, provides administrative GUI for DSCMS management and at-a-glance views of DSCMS operational status such as registered users, current log-on users, display device status, and so on. Web-based dashboard is popular since it allows access to DSCMS server from any place through Internet.

※ Corresponding Author : Sun-Tae Chung, Address: (06978) Dept. of Smart Systems Software, Soongsil Univ., 369, Sangdo-Ro, Dongjak-Gu, Seoul, Korea, TEL : +82- 2-820-0638, FAX : +82-2-821-7653, E-mail : cst@ssu.ac.kr
Receipt date : Feb 15, 2017, Approval date : April. 28, 2017

[†] Dept. of Information and Telecommunication Engineering, Soongsil University
(E-mail :{khue.trinh, doublestat, kesuskim}@ssu.ac.kr)
[††] Embedded Vision, Inc, Seoul, Korea
(E-mail : binh.nguyen@ieev.org)
[†††] Dept. of Smart Systems Software, Soongsil Uniersity
(E-mail : cst@ssu.ac.kr)

Most of conventional DSCMS's with web-based dashboard have been developed based on LAMP/WAMP platform (Linux/Windows, Apache Web Server, MySQL, PHP) or variants [3,4], with AJAX-based proprietary communication protocol and/or SOAP-based web service.

In web-based applications, REST API [5] becomes a popular web service mechanism between servers and clients since it has cleaner and easier interface over non REST HTTP API and lighter than SOAP-based web services. Under traditional AJAX-based or SOAP-based networking environments, it is well known that real-time networking(messaging) is difficult to implement efficiently.

Recently, MEAN stack (a SW bundle of Mongo DB, Express.js, AngularJS and Node.js)[6] or MERN stack (a SW bundle of MongoDB, Express.js, ReactJS and Node.js) [7] has been widely adopted platform for developing web applications; MongoDB [8], Express.js[9], and Node.js[10] for server side, and AngularJS[11] or ReactJS[12] for client side. MEAN or MERN stack-based web application development increasingly gains popularity for several advantages; fast performing and flexible web server architecture (Node.js, Express.js), easy to support real-time communication (asynchronous event driven processing of Node.js, Socket.IO library [13]), flexible and easily scalable DB (MongoDB), clean and maintainable front-end design (AngularJS, ReactJS), the same development language, JavaScript and the same data format, JSON [14] both on server side and client side, and convenient development environments. JSON (Java Script Object Notation) is the format for data-interchange through all the layers. Even though XML may have more representational power, JSON has also good enough representation capability and can be parsed by a standard JavaScript function. MongoDB stores data records as JSON documents. Moreover, JSON also allows working with external APIs easily. MongoDB is superior for distributed databases and is highly scalable.

Under MEAN or MERN stack platform, REST API is simple and easy to implement with JSON as data format. Thus, REST API suits MEAN or MERN stack environments well.

Developing a real-time applications with popular web application stacks like LAMP (PHP) has traditionally been very hard. It involves polling the server for changes, keeping track of timestamps, and it's a lot slower than it should be. Recently, Socket.IO[13] has been the solution around which most real-time communication systems under web environments are architected, providing a bi-directional communication channel between a client and a server. Whenever an event occurs, the idea is that the server or clients will get it and push it to concerned connected clients or the server. In Node.js environments, Socket.IO is implemented as Node.js module, and thus is well supported under MEAN or MERN stack.

In this paper, we propose a web-based real-time digital signage system, MR-DSS, based on MEARN stack under cloud computing environments, with a target for a signage service platform, not a signage solution. Here, MEARN stack implies that the CMS server of the proposed MR-DSS, is constructed based on Node.js, Express.js and Mongo DB. Dashboard of MR-DSCMS, and signage player are built on AngularJS and ReactJS, respectively. Dashboard needs to provide an administrative GUI for user management, presentation management, schedule management, asset management, device (displayer) management, status of DSS, statistics of audience response, and so on. All GUI views are shown in comprehensively and systematically, and Some GUI views are supposed to be shown differently depending on user level (privilege level), and an user may sometimes choose to see some views only in large-view mode. Therefore, more modular and flexible design and architecture are desirable. AngularJS is a front-end framework on which web GUI design with more modular and flexible architecture can be built. ReactJS is not a framework

but a convenient, flexile, light library for supporting rendering front-end display quickly. For signage displayer, which does not need complicated GUI, but needs a fast but flexible display, ReactJS fits signage player display more efficient than AngularJS since AngularJS is heavier and more complicated.

MR-DSS not only supports normal main CMS services via REST APIs, but also supports real-time event processing task by utilizing Socket.IO. The efficient implementation of REST APIs under MEARN stack in MR-DSS is shown through comparison experiments with Xibo DSS, which is a popular open source DSS. Also, real-time service implemented in MR-DSS based on Socket.IO is shown to be faster compared non real-time service, REST APIs.

Even though there appear many WCMS(CMS for web site) based on Node.js [15,16], there are a few WCMS based on MEAN/MERN full stack [17]. To the best of our knowledge, no digital signage CMS based on MEARN stack with support of real-time event processing has neither been reported in the literature and has nor been commercially released.

The rest of the paper is organized as follows. Section 2 describes related work necessary for understanding the paper. Section 3 discusses our proposed architecture. Section 4 shows the design of issues and implementation. Some evaluation is presented in Section 5. Finally, the conclusion is given in Section 6.

2. TECHNICAL BACKGROUNDS AND RELATED WORK

2.1 Technical Backgrounds; Node.js, MEAN/MERN stack, REST API, and Socket.IO

A. Node.js [10]

Node.js is an open-source, cross-platform JavaScript runtime environment for developing a diverse variety of applications. Node.js has an event-driven architecture capable of asynchronous I/O, which makes Node.js excel at multi-user, real-time web applications. The real-time power comes through use of the WebSocket protocol supported by Socket.IO library. Research has shown that a Node server significantly outperforms both Apache and Nginx [18] for serving dynamic content—and Node's implementation of JavaScript is more than 2.5 times faster than the more traditional PHP approach, by efficiently utilizing available hardware.

Node.js is now a popularly adopted web application framework and starts to be accepted as a good platform for CMS [15,16,17] since it is a concurrent JavaScript environment for building scalable and fast web applications.

B. MEAN/MERN/MEARN Stack

MEAN stack [6] means a S/W bundle of MongoDB, Express.js, and Node.js for server-side and AngularJS for client-side, which is well-known for building dynamic web sites and web applications. MERN stack [7] is the same as MEAN except that AngularJS is replaced by ReactJS. MEARN stack in this paper means that both of AngularJS and ReactJS are used for client-side development in addition to MongoDB, Express.js, and Node.js for server-side.

- Express.js[9] is built on the underlying capability of Node.js and provides a web application server framework which supports handling routing and HTTP operations (such as GET and POST) conveniently. Express.js facilitates a simplified and more elegant solution than (re-)implementing these services directly using Node.

- MongoDB[8] is well-known for a high performance, well-scalable document-oriented NoSQL database built around the JSON data format [14] and as such is ideally suited to a server-side JavaScript environments such as those provided by Node.js.

- AngularJS[11] is a frontend JavaScript frame-

work to develop complex client side applications with modular code and data binding UI. Angular implements the MVC pattern to separate presentation, data, and logic components. Current AngularJS version is Angular2, which has a new architecture different from Angular1.

- ReactJS[12] is a declarative, efficient, and flexible JavaScript library for building user interfaces. React creates an in-memory 'virtual DOM', computes the resulting differences from real DOM, and then updates the browser's displayed DOM efficiently. Thus, ReactJS can support rendering views quickly. As opposed to MVC pattern adopted in AngularJS, ReactJS utilizes flux pattern, which is unidirectional so that it does not cause confusing and it defines workflow clearly. Redux is a very lightweight (2kB) implementation of flux. ReactJS has smaller size and faster processing speed than AngularJS since it is library and uses virtual DOM.

C. REST API [5]

REST stands for Representational State Transfer. REST API, widely known as RESTful API or RESTful model for web-services, uses the native HTTP operations: POST, GET, PUT & DELETE to map on to the four fundamental CRUD operations (Create, Read, Update & Delete) on resources, represented by a URL. The URL describes the object to act upon and the server replies with a result code and valid JavaScript Object Notation (JSON). Because the server replies with JSON, it makes the MEARN stack particularly well suited for our application, as all the components are in JavaScript and MongoDB interacts well with JSON.

REST API is simple and easy to implement compared to SOAP, which has been a facto standard for web services.

D. Socket.IO [13]

Socket.IO is a JavaScript library for real-time web applications. It enables real-time, bi-directional communication between web clients and servers. It has two parts: a client-side library that runs in the browser or clients, and a server-side library for Node.js. Both components have a nearly identical API. Like Node.js, it is event-driven. Socket.IO primarily uses the WebSocket protocol with polling as a fallback option, while providing the same interface.

E. SMIL Timesheets

Timesheets[19] defines declarative W3C standards based on SMIL Timing [20] and SMIL Timesheets[21] to synchronize HTML contents. SMIL Timing defines elements and attributes to coordinate and synchronize the presentation of media over time.

Basically, SMIL Timing allows an a-temporal language such as HTML5 to be extended with timing features. SMIL Timesheets reuses a significant subset of SMIL Timing and allows timing and synchronization to be separated from content and presentation. This can be seen as the counterpart of CSS style sheets in the timing domain: like in CSS, these features are gathered either in an external resource linked to the document, or in a timesheet element in the document itself. Like CSS style sheets, timesheets can be associated not only to HTML pages but also to other types of documents such as SVG drawings, for instance, or even to compound documents made of HTML and SVG.

Based on the observations above, it appears that combining HTML5+CSS3 and SMIL Timing would bring a good solution to support temporal synchronization as well as spatial synchronization among assets in a presentation.

2.2 Related Works

Content management system (CMS) is a computer application which provides an efficient content management tasks. The content management tasks include storage, retrieval, creation, edition,

organization, distribution of contents [22], access control to contents, other additional tasks depending upon usage purposes.

Digital Signage CMS(DSCMS) is different from WCMS (Web CMS), which is responsible for content management in a web site. DSCMS supports scheduling of content dissemination so that they can be displayed at a desired time while WCMS supports organization of contents in a web site in a way that the contents are wanted to be shown when people visit the web site. Most of signage content management systems also support content creation and edition by a subsidiary content design module or program [15,16,17], as well as user management, device management, asset management, and so on.

Dashboard of DSS, which is a front-end administrative GUI to back-end CMS server, provides at-a-glance views of DSCMS operational status such as registered users, current log-on users, user privilege levels, display device status (connected, disconnected, live, dead), contents displaying on display devices, network health reports, screen GEO maps, communication between users, audience measurement statistics as well as management GUI (user management, schedule management, and more). The dashboard is usually constructed in web base so that it is displayed on a web browser.

In order to support interoperability among digital signage products and services from different manufacturers, operators, and a wide variety of terminals (display devices) with different characteristics, several industry initiatives such as POPAI [23], Intel[24], Digital Signage Federation[25], and standardized organization such as ITU-T[26], W3C Business Group [27,28], have been formulated to define technical specifications for interoperable digital signage solutions. However, those standard efforts have not kept pace with flourishing deployment of commercial or open source digital signage systems.

Many of conventional digital signage CMSs [3,4] have been developed based on LAMP platform (Linux, Apache Web Server, MySQL, PHP) or variants; WAMP(Windows, Apache Web Server, MySQL, PHP), LAPP(Linux, Apache, PostgreSQL, PHP), MAMP (Mac OS X, Apache, MySQL, PHP), XAMPP (Linux, Mac OS X, Windows, Apache, MySQL, PHP, Perl). and many of them have utilized their own contents description in XML or JSON.

Nowadays, due to the efficiency and simplicity of the MEAN/MERN full stack platform, it has been popularly adopted for developing web-driven applications including WCMS and digital signage CMS [15,16,17].

In almost all digital signage systems, currently content representation about spatial synchronization among multimedia assets like text, image, and video is usually described declaratively, either in HTML, or in XML, or JSON. On the other hand, content control including temporal synchronization among assets can be achieved by either imperative or declarative approach. The imperative approach involves JavaScript codes which are programmed about how to control contents and playout. In the declarative approach, a mark-up or a format description annotates control of contents. Then, signage players parse the mark-up or format description and display contents.

The declarative approach is generic so that it is useful for not only developers but also vendors of authoring tools. If the annotation format is standardized, anyone could develop JavaScript libraries for playing signage contents. This means that the declarative approach could make signage operations more cost-effective. Furthermore, this could achieve interoperability among terminals using ordinary web browser.

Many declarative approaches have been proposed; HTML, SMIL[29], SCXML[30], Timesheets [19]. Timesheets approach is based on SMIL Timing [20] and SMIL TimeSheet[21]. SMIL Tim-

esheets is a style sheet language which is intended for use as an external timing stylesheet for the Synchronized Multimedia Integration Language, and is meant to separate the timing and presentation from the content inside the markup of another language (for instance, an SMIL Timesheet can be used to time an SMIL-enabled slideshow).

We adopts Timesheets approach since we can use HTML/CSS and JSON as they are and supports timing control by additional mark-ups or JSON formats which can be interpreted by a JavaScript library.

3. THE PROPOSED MEARN STACK-BASED REAL-TIME DIGITAL SIGNAGE (MR-DSS)

3.1 Design Principles

The proposed digital signage system, MR-DSS has the following requirements;

1) MR-DSS is structured as a web-based application.

- Web-based digital signage system is built on base of well-accepted web technologies; http, JavaScript, JSON, and others.

- MEAN/MERN stack is a useful and convenient framework for constructing web-based application. Thus, we construct the digital signage system, MR-DSS, based on well-accepted web technologies and MEARN stack.

2) MR-DSS supports non real-time information processing through REST API

- Signage players or dashboard contacts MR-DSCMS (CMS of MR-DSS) server to receive signage presentation information, and management information, respectively. The request and reception of these information do not have to be processed in real-time, In this case, it is more appropriate to support them by pull style non real-time communication like HTTP. REST APIs over HTTP are simpler than SOAP, and fits well for MEAN/MERN stack-based web applications.

3) MR-DSS supports real-time information processing from or to CMS server through Socket.IO.

- Digital signage systems need to support distribution of urgent information like disaster messages and monitoring the status of signage players. For these purpose, polling using HTTP isn't appropriate because it needs a certain amount of time to processing information over networking. Even though web supports many mechanisms like WebSocket, server-sent events, and push API, we construct real-time processing based on Socket.IO, a JavaScript library based on WebSocket.

4) Support the representation and control of signage contents through declarative approach

- As explained in Section 2.2, as for content control including temporal synchronization among assets in a content frame can be achieved more generically by declarative approach. The proposed digital signage system, MR-DSS adopts time synchronization description based on Timesheets approach [19]. Thus, our description about content description consists of two parts; JSON description for spatial layout, and JSON format for temporal control description based on TimeSheets.

3.2 System Working Architecture

The proposed MEARN Stack-based real-time digital signage system (MR-DSS) as the following working architecture as shown in Fig. 1.

The proposed signage system, MR-DSS has two main components: MEARN stack-based real-time digital signage CMS (MR-DSCMS) located on a cloud network and signage content players (SCPs) at PC or embedded systems. MR-DSCMS consists of front-end dashboard clients and a main processing server (MR-DSCMS Server) based on MEARN stack.

The dashboard clients provide administrative

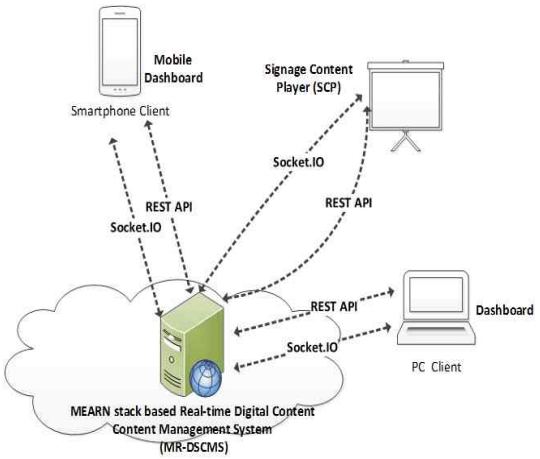


Fig. 1. System Working Architecture.

web GUI to MR-DSCMS. Mobile dashboards on Android or iOS are limited version of PC dashboard and constructed based on native systems, not like PC dashboard, which is built based on AngularJS JavaScript framework. SCP renders signage contents described in JSON format by utilizing React JS.

MR-DSS supports two kinds of communication mechanisms. For loosely coupling with RESTfulness and openness, non real-time communications between MR-DSCMS server & SCP, and MR-DSCMS server & dashboard are processed by REST APIs. For real-time tasks such as real-time monitoring of signage players' status, distribution of urgent messages, real-time processing of commands, notification of new schedules and contents, and sending of instant messages are processed in real-time via Socket.IO protocol.

Signage content presentation, which means a content frame to be displayed on SCP at a time, is designed and formatted in JSON from the layout editor in MR-DSCMS or manually. It can be also designed in HTML format from HTML designer like google web designer. It can be redesigned from editing pre-existing content presentations. When a new schedule for new contents is ready, then MR-DSCMS server notifies the fact to related SCPs via Socket.IO. Then, the SCPs contact MR-DSCMS server and download the schedule/playlist and the edited presentations from the MR-DSCMS server via REST API and render them according to schedule or playlist.

A signage content presentation has a spatial layout and temporal control among constituents (assets) in the layout. The layout consists of many asset types such as a text, image, graphic, video, or widget. A widget is a small JavaScript codes with assets or links to assets, which can do some useful tasks (like showing weather information). Spatial layout and temporal control in rendering presentation are described mainly in JSON format and description by HTML5/CSS3 + TimeSheets format is allowed in the sense that it can be inserted in JSON format files. A playlist is a list of presentations to be displayed with durations on SCP. A schedule (file) is a document which has information for assigning start time and playing duration, and display days in a week to each presentation. Playlist and schedule are all described in JSON format.

The proposed MR-DSS supports monitoring what signage content a specific SCP is playing out through webRTC[31], which will be explained in detail in Section 4.8.

3.2 MR-DSCMS Server Architecture

MR-DSCMS is designed to consist of MR-DSCMS Server and administrative web GUI client, dashboard, which is loosely decoupled from MR-DSCMS Server in the sense that they are constructed as client/server based on MEARN stack. Fig. 2 shows overall S/W architecture of MR-DSCMS Server.

MR-DSCMS Server is designed to have three-tier architecture (front-end layer (Nginx server), application layer (MR-DSCMS application processing), data layer (database) based on MEAN stack.

Front-end Nginx server plays a role of load balancing. Application layer is responsible for han-

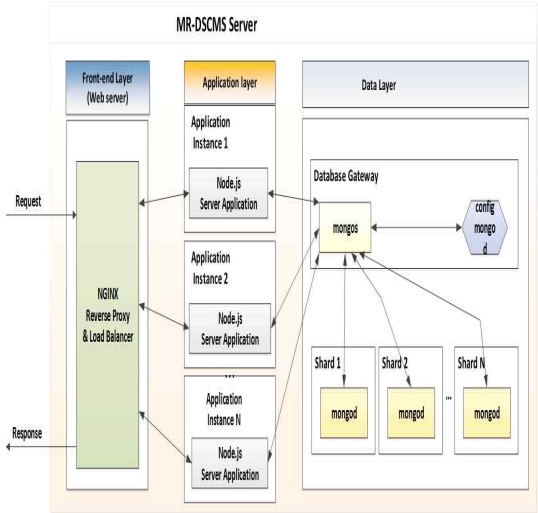


Fig. 2. S/W Architecture of MR-DSCMS Server.

dling business logic about user, schedule, presentation, device, asset management, and etc., and supports scalability to adapt traffics by increasing Node.js application instances according to traffics. The architecture of it is clarified in Section 3.4. Data layer handles data access to a database (MongoDB).

3.3 MR-DSCMS Server Data Layer

MR-DSCMS supports to manage user, asset, device, schedule, presentation, system administration of MR-DSS, and so on. Thus, current MongoDB is designed to consist of the following collections; User, Device, Presentation, Asset, Weekly Schedule, Real-time Schedule, Playlist, Display Event, System Config, ECoin Transaction, Device Status, User Logs, System Logs, Event Logs, Sensor Status, Actuator Status, Audience Response, and so on. The System Logs, and User Logs are used to save history information of system and user, and all these log information of more than 30 days old will be cleaned automatically if no other settings are not designated.

We use mongoose as Object Document Mapper (ODM) for MongoDB. Through sharding, scalability of MongoDB is supported. Mongod is the primary daemon process for the MongoDB system. It handles data requests, manages data access, and performs background management operations. Mongos for "MongoDB Shard," is a routing service for MongoDB shard configurations that processes queries from the application layer, and determines the location of this data in the sharded cluster, in order to complete these operations. From the perspective of the application, a mongos instance behaves identically to any other MongoDB instance.

3.4 MR-DSCMS Server Application Layer Architecture

Fig. 3 shows the S/W architecture for MR-DSCMS application layer of MR-DSCMS.

The MR-DSCMS server's application layer is constructed based on Node.js/Express.js and MongoDB is utilized as a database. The server also takes responsibility for managing contents, users, device players, schedules comprehensively and consistently. The server's management services are designed to be provided to clients by REST

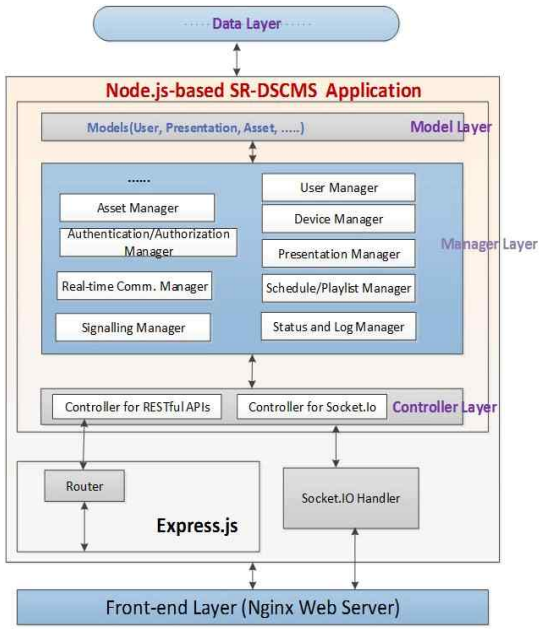


Fig. 3. Node.js-based MR-DSCMS Application Layer Architecture.

APIs with the corresponding data stored on a MongoDB database.

When a client sends a request using a REST API to MR-DSCMS server, the router modules in application layer will route and dispatch it to the appropriate controller. Authentication/Authorization manager handles the authentication and authorization by using JWT (JSON Web Token). A module in Controller layer processes the REST API and, ask the corresponding manager in Manager layer for dealing with the request task appropriately. Socket.IO handler does in similar way to router modules and dispatches a real-time communication request/response via on Socket.IO, to the corresponding controller module.

Modules in Manager layer are responsible for business logics handling nonreal-time management tasks such as user management, device management, and so on, and real-time management tasks (event handling, etc.). For scalable extension, another instance of application layer based on a Node.js/Express can be launched on a processing unit like CPU.

3.5 Signage Content Player (SCP) Architecture

ReactJS is smaller and faster than AngularJS. And, it is a library not a framework like AngularJS, so that it can be easier to be integrated with other libraries. The S/W architecture of Signage Content Player (SCP) is illustrated in Fig. 4.

After SCP downloads a presentation description in JSON format from MR-DSCMS server using REST API or finds it from local storage, the presentation manager will parse the presentation JSON description, and then the display manager will construct virtual DOM using ReactJS API based on parsing presentation information, and render using ReactJS API into connected display device in the Display Component process.

The Monitoring Service handles real-time SCP status (live or dead) update, gathering information such as playing mode, playing content type (presentation, playlist or schedule), the location, memory use, collection of surround sensors' data, audience responses, and then report them to MR-DSCMS server via Socket.IO. The screen of display device will also be captured and video will be streamed to MR-DSCMS using singnalling and-web RTC in this process, which will be explained in more detail in Section 4.7.

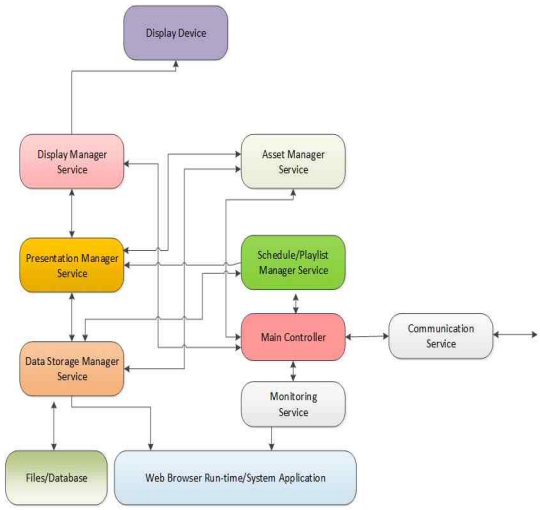


Fig. 4. S/W Architecture of Signage Content Player.

Currently, we implemented Android version as well as PC version. SCP on PC is constructed based on electron platform [32], Android version implemented above architecture using native android framework, and is being migrated into React-based architecture using Reactive native.

3.6 Dashboard Architecture

The architecture of MR-DSCMS Dashboard is illustrated in Fig. 5, which are constructed on Angular2 framework. Basically, components with views in component layer are designed respectively for each dashboard administrative work; user management, presentation management, schedule management, device management, layout editor, statistics, log, system management and so on. Layout editor component utilizes Fabric.JS, a Javascript HTML5 canvas library. A content pre-

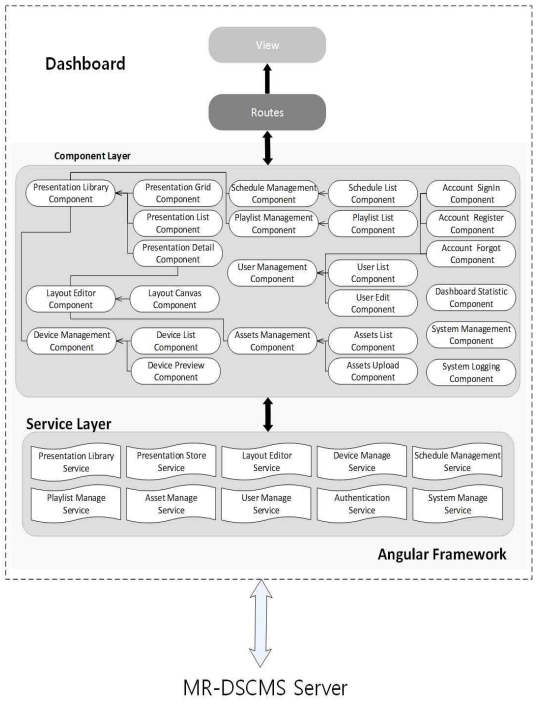


Fig. 5. S/W Architecture of Dashboard.

sentation edited by layout editor component is saved in JSON format and saved into MR-DSCMS server. Services at service layer provides REST API handling with MR-DSCMS server for each corresponding component.

4. Design Issues and Implementation of MR-DSCMS

4.1 REST API Design

REST uses a client-server model, where the server is an HTTP server and the client sends HTTP verbs (GET, POST, PUT, DELETE), along with a URL and variable parameters that are URL-encoded. The URL describes the object to act upon and the server replies with a result code and valid JavaScript Object Notation (JSON) [14].

Because the server replies with JSON, it makes the MEARN stack particularly well suited for our application, as all the components are in JavaScript and MongoDB interacts well with JSON. The CRUD(CREATE, READ, UPDATE, and DELETE) acronym is often used to describe database operations. These database operations map very nicely to the HTTP verbs; POST(insert or create data into server), GET(retrieve data from server), PUT(update the data into server), DELETE(remove data from server).

The REST HTTP API in this paper is structured as the following format; '{METHOD}' 'https://API domain/prefix/collection name/extra functions'

- Method: GET, POST, PUT, DELETE
- API domain: in our example, we use 'api.erclab_ssu.com'
- Prefix: • [auth] if need authentication
  • Don't have prefix if allow directly access without authentication
- Collection name: the name of collection of MongoDB which we want to deal with.
- Extra function: if the API is not basic CRUD, it will need this.

Examples of API format are as follows.

- Login; {POST} http://api.erclabs_ssu.com/log-in
- Get info of an asset; {GET} http://api.erclabs_ssu.com/auth/asset
- Create an asset; {POST} http://api.erclabs_ssu.com/auth/asset
- Update user information; {PUT} http://api.erclabs_ssu.com/auth/user

4.2 Real-time Service Implementation based on Socket.IO in MEARN platform

In time-sensitive information processing, polling using HTTP isn't appropriate because it needs a certain amount of time to get information. W3C recommends the following APIs for real-time communication for digital signage system: Web Socket API, Server-Sent Events, Push API.

In the proposed system, we support real-time communication by 'publish/subscribe' mechanism based on Socket.IO library. When a client or CMS server wants to receive real-time messages about some events (disaster message, instant message,

notification message about new schedule/playlist, signage player status message, sensor status message, and etc.), then it calls 'subscribe' method with the event type and a callback function. The 'subscribe' method first enrolls the event and callback function into the 'subscribe' management data structure, and sends the event type to the other party (CMS server or client). The other party enrolls it in the event receiver list. When the event happens, then the other party calls for 'publish' method which looks for receivers of the event, and sends the event message to receivers one by one using Socket.IO function. When the event message arrives, the callback function registered to 'subscribe management data structure' is called and handles the event message.

4.3 Authentication and Authorization

Authentication means the ability of the server to determine the identity of the clients in a transaction. Authorization means allowing or preventing clients to use a particular resource or service by the server. For the proposed system, JWT (JSON Web Token) is utilized to authentication and authorization since it is simple and efficient. JSON Web Token (JWT) is a JSON-based open standard (RFC 7519) for creating access tokens that assert some number of claims. As opposed to session cookie and OAuth2 protocol [33], JWT is stateless in the sense that session information or user state is saved in server memory. And it can include additional claims such as the users email address, who issued the token, scopes or permissions for the user, and more while session cookie contains only session id. Compared to OAuth2, JWT is simple since it is just protocol, and it is not a security framework like OAuth2. With a cookie based approach, you simply store the session id in a cookie. JWT's on the other hand allow you to store any type of metadata, as long as it's valid JSON. Depending on your use case, you may choose to make the minimal amount of claims such as the user id and expiration of the JWT, or you may decide to include additional claims such as the users email address, who issued the token, scopes or permissions for the user, and more. The JWT in the proposed system is designed to contain user id, user level, issue time, life time. The life time of JWT can be manually configured by the administrator (Super-admin).

As for authentication of REST HTTP API request, it works as follows. After a user logins, then the MR-DSCMS server validates the user on the backend by querying in the database. If the login request is valid, the MR-DSCMS server creates a JWT token by using the user information fetched from the database, and then return that information in the response header so that the user can store the token in local storage. Once the client has that token, they will store it in local storage. This token is passed at the authorization header in every REST HTTP API request. The JWT token accompanied with REST HTTP API request is validated in 'validateRequest' middleware module in Express.js.

As for authorization of REST HTTP API request about how much the API has privilege for accessing resources including system information, files, database information like asset, the information extracted from JWT such as user id, user level, and others are utilized.

4.4 User Management

User management handles user registration, login, and user level. User level is related with access privilege to APIs, assets, and what each user can see and what menus he is allowed to use. According to user's role, each user will have different user level. The proposed MR-DSS supports three level groups of users, which are stored in "role" field of User model database; Admin group (Super-Admin, Admin, Moderator), User group (Agent, End-user, Anonymous), and Guest. Super-Admin and Admin can manage everything, but

Admin can be many and added or removed. On the other hand, Super-Admin has root privilege and it can manage everything, and never be eliminated. Moderator can manage information except for system information.

Each end-user associates at least a SCP with it. SCP is only allowed to be associated with to a user. Each end-user group can be represented by an agent. An end-user can belong to a group managed by an agent by declaring an associated agent. All user management is designed to be processed in MR-DSCMS. User is related to authentication, authorization, and device (SCP) management. Based on the role of user, the system will decide to give the permission to execute a function or not.

The proposed system is designed also to allow to log in through SNS like Facebook, google, and twitter.

4.5 Presentation Management

A presentation is described by a JSON document. The presentation JSON document describes not only spatial layout of a content screen and but also temporal synchronization among regions and among assets in a region. Temporal synchronization allows slideshows, animation of images in a region.

The presentation description contains two mandatory main parts: meta, regions, and one optional part; timing. A presentation can have one or many regions. Meta part of a presentation, denoted by Meta_P in JSON documents, keeps the attributes of presentation such as the presentation id, presentation name, owner, status, onStore. creation/modification time-stamp, type, size, resource URL, category, usage level, lock property, and the orientation of display. Lock property determines whether the presentation can be edited by the group of owner or buyers except the owner. Status represents activation; activated(default), deactivated. Onstore attribute shows whether the presentation belongs to store or not. Timing part of a presentation, denoted by Timing_P in JSON documents, has description about temporal synchronization among regions based on SMIL Timesheets if any.

Similarly, each region contains two mandatory parts; meta and assets, and one optional timing part. Each region can be composed of one or many assets of the same type. Meta part of region (denoted by Meta_R) keeps the attributes of the region such as the region id, creation/modification time-stamp, asset type, size (width, height), position (x,y) in the layout, z-order, asset list, usage level, and lock property. Asset list keeps the number of assets, asset ids. Asset part has also meta part and data part. Meta part of asset, denoted by Meta_a, has attributes like asset id, asset name, asset type, asset size, asset file extension, and so on. Data part of the asset has a text string for text type and file id or URL if asset types are image or video. Timing part of a region, denoted by Timing_R in JSON documents, has description about temporal synchronization among assets based on SMIL Timesheets if any. Fig. 6 shows

```
Regions: [
    {
      Meta_R : {
        type: 'image', // or 'text', 'video', 'widget', 'webpage', ...
        id: <REGION-UID>,
          ...
      Asset_list : {
          total number : 2 ,
          asset_id: '5829b0c48e15ca3dc0cb49e2',
          asset_id: '5829b0c48e15ca3dc0cb49e3'
              } },
      Assets: [
        { //1st asset
          Meta_a : {
            id: '5829b0c48e15ca3dc0cb49e2',
            type: 'image',
            name: 'Mcdonalds Logo',
            ...
                }
          data: {
            files_id: '5829b0c48e15ca3dc0cb49e1'
              }
        },
        { //2nd asset
          Meta_a : {
            id: '5829b0c48e15ca3dc0cb49e3',
            type: 'image',
            name: 'Lunch Menu',
            ...
                }
        data: {
            url: 'https://example.org/lunch_menu.png'
              }
          }
        ]
      Timing_R : {
        Seq :  {
          repeatCount : 'indefinite'       },
        item : {
        select : '#Asset_list img' ,
        dur : 3  //3 second          }
        },
      ...
    }
  ]
```

Fig. 6. JSON description example for regions in a presentation.

a JSON description example for regions in a presentation which has image assets and optional timing part about how to animate images.

4.6 Schedule/Playlist/Display Management

The playlist lists presentations to be displayed with durations in a SCP. There is two types of schedule: weekly and real-time schedule. The weekly schedule file that assigns start time, playing duration, and playing days in a week to each presentation. The real-time schedule file that assigns flexible start time information as a calendar. The schedule contains meta information about presentations, name of presentation, days in week to display, start time, and playing duration time.

The information about playlist and schedule contents are kept in the corresponding collections in database. MR-DSCMS allows users to edit schedule/playlists for associated display devices (SCP). The updated schedule/playlist can be different from the original schedule/playlist in contents. If the updated ones can be still applied to the same display devices, then the original schedule/playlist can keep the same ID except different presentations. However, if the updated schedule/playlist has different applicable display devices (SCPs), the updated schedule/playlist will be assigned a new ID and the original ones will be assigned a new ID by Schedule/Playlist Manager in order to keep consistent management of schedule/playlists.

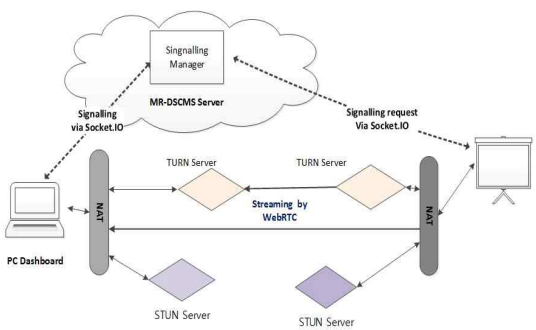


Fig. 7. Signalling and video streaming architecture between dashboard and SCP.

There might be a time conflict between event driven contents and scheduled contents. Recently, digital signages are developed to be interactive so that they can provide additional information (event driven contents) in response to user inputs. Advertisers want a signage to display their paid advertisements (scheduled contents) on time. Advertisement is a major funding source for digital signages. Therefore, it is required to handle this issue. In SMIL 3.0, there is a discussion of unifying event based and scheduled timing for multimedia presentation. Currently, we are working for the unification.

4.7 System Monitoring

Each SCP on the system always keep a connection with MR-DSCMS for reporting real-time status update. SCP will send a real-time updated status message through 'publish/subscribe' mechanism (constructed based on Socket.IO) to MR-DSCMS at start time, and periodically during operation and when there is any status change. This status information will be kept in corresponding models (DeviceStatus, Sensorstatus, EventLogs, SensorStatus, ActuatorStatus, AudienceResponse), which are used to diagnose and analyze system status, audience response for system monitoring. Then, MR-DSCMS provides relevant users the necessary statistical data, such as weekly system status report or audience response report.

SCP also supports capturing a screen shot. When a user wants to see what is playing on a specific SCP in entire network, from the dashboard, he requests a signaling via Socket.IO to the signaling manager in CMS server. Then, singnalling manager sends asking for signaling information to the SCP. Then, the singnalling manager sends back video channel connection between dashboard and the SCP, which has been obtained from outside STUN server, to each dashboard and the SCP. Then, the dashboard and SCP establishes webRTC

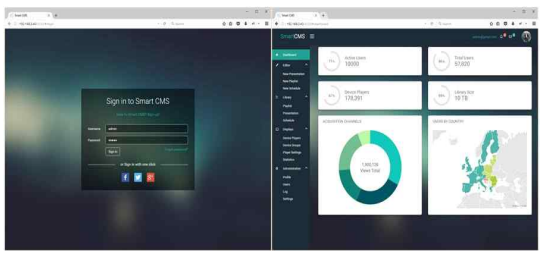
video session channel between two. Then, the SCP sends the screen shot captured by WebRTC Tab Content Capture API to the dashboard via webRTC channel.

5. EXPERIMENTS

5.1 Evaluation of System Operations

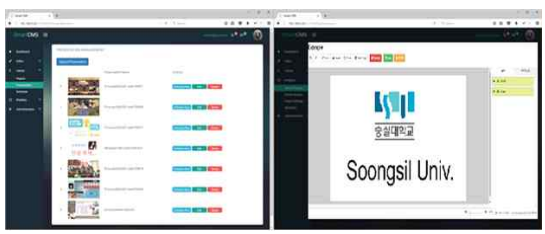
We tested and evaluated our implementation of the proposed system. Currently, we mainly concentrated on testing of managing MR-DSCMS and working between MR-DSCMS located in Google Cloud and SCP. Two components can communicate each other through Internet. Fig. 8 shows log-in GUI (left) and home administrator GUI (right) of MR-DSCMS's dashboard GUI respectively. After logging on, end users see home administrator GUI which shows the statistical data analysis.

Dashboard supports administrative managements, which are shown in the left frame in Fig. 8(b). Fig. 9(a) shows presentation lists after one selects 'presentation' menu in the left frame in Fig.



(a) Log-in GUI (b) HomeGUI after Log-on

Fig. 8. (a) Log-in GUI, (b) Home of MR-DSCMS's Dashboard.



(a) Presentation lists (b) Layout editor

Fig. 9. Presentation lists and Layout Editor GUI.

8(b). Fig. 9(b) shows Layout editor GUI.

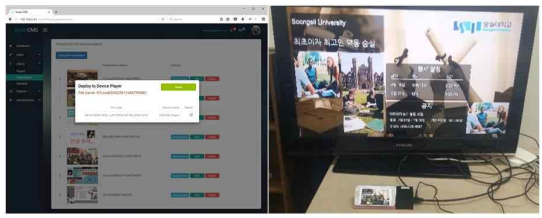


Fig. 10. Dispatching scheduled presentation for signage content players and working environment.

Dispatching a scheduled signage content to signage content players and working environment (Smartphone, set-top-box, monitor) are shown in Fig. 10.

Through lots of testing, it is evaluated that all of tried users feel editing the contents (presentations) and dispatching them into signage content platers to be done conveniently and smoothly and observer engaged SCPs to play the presentations according to schedules well.

5.2 Evaluation of Networking Load Performance

5.2.1 Performance Evaluation of Response time and Throughput of REST APIs

For the load performance evaluation of the proposed MR-DSCMS, we tested response times of REST APIs of MR-DSCMS and compared them with those of REST CMS APIs of a famous open source LAMP-based DSS, Xibo (version 1.8) [3]. Xibo supports two types of APIs; Player APIs and CMS APIs. Player APIs are for exchange of information about content display and display statistics between Xibo CMS and Xibo Players. The Player APIs are constructed based on SOAP service. Xibo CMS APIs[34] are for providing CMS management information and are constructed as REST APIs under LAMP environments. Functionalities provided by either soap-based Player APIs or REST-based CMS APIs of Xibo are done by REST APIs in the proposed MR-DSCMS. We did not test the response times of the SOAP-based CMS APIs since SOAP is well known to be slower

than REST APIs[34]. Comparison with other commercial DSS's with respect to performance was not possible due to unavailability of them. We installed the proposed MR-DSCMS and Xibo CMS in Google cloud platform (Google Compute Engine) under the same computing environments; Machine Type: g1-small, CPU: 0.5 vCPU Intel Zeon 2.6 GHz Intel Xeon E5 (Sandy Bridge), RAM: 1.7GB.

For fair comparison, we chose five APIs with similar functionalities from MR-DSCMS and Xibo-CMS [35]. Measurements have done by Apache JMeter tool [36] configured with 1000 samples (request numbers) where each REST API is called. Table 1 and Table 2 summarize experimental results with JMeter. In Tables, 'Samples' denotes the number of REST API requests ran for given thread, and 'Average' means the average response time in millisecond for that particular REST API, "Throughput' means the number of requests per second, that are sent to MR-DSCMS or Xibo-CMS during the test, and 'Avg. Bytes' means the average size of the sample response in bytes.

Experimental results of Table 1 and Table 2 show that overall, the REST APIs of the proposed MR-DSCMS performs faster and produces more throughput than those of the open source Xibo-CMS. These experimental results comes from the fact that Node.js/Express.js framework is lighter and more efficient in handling requests with light processing than Apache/PHP framework and that processing REST APIs on MEAN stack is simpler and easily than that on LAMP stack.

5.2.2 Performance Evaluation of Real-time Communication Service based on Socket.IO

In digital signage systems, CMS needs to send real-time messages like instant message to players, and a player needs to send its status information to CMS in real-time [28]. In the proposed MR-DSS, real-time messaging service is supported via on Socket.IO. As for how commercial DSS's support real-time messaging service is not publicly exposed. Open source Xibo DSS supports a push message mechanism called XMR(Xibo Message Relay) based on a messaging queueing service, ZeroMQ[36] for sending updated schedules

Table 1. Experimental results of Network load test about REST APIs of MR-DSCMS

APIs	Avg. (ms)	Min (ms)	Max (ms)	Std Dev	Throughtput (req/sec)	Avg Byte
Players	70	63	171	13.97	22.9/sec	795.0
Users	71	64	143	10.83	22.9/sec	245.0
Presentations	69	62	140	11.94	22.9/sec	801.0
Schedules	70	63	142	12.71	22.9/sec	803.0
Assets	69	63	139	11.00	22.9/sec	798.0
Overall	69.8	63	171	12.09	22.9/sec	688.4

Table 2. Experimental results of Network load test about REST APIs of Xibo-DSCMS

APIs	Avg (ms)	Min (ms)	Max (ms)	Std Dev	Throughtput (req/sec)	Avg Byte
Displays	756	108	4424	546.07	2.6/sec	346.3
Users	782	99	5026	593.95	2.6/sec	1181.3
Layouts	744	104	4517	535.95	2.6/sec	1091.3
Schedules	740	106	4267	526.39	2.6/sec	227.3
Medias	758	102	6261	551.24	2.6/sec	4045.3
Overall	756	99	6261	550.68	2.6/sec	1378.3

or presentations to players. ZeroMQ is constructed directly on the top of TCP. On the other hand, Socket.IO is constructed on the top of HTTP, and thus it is an application layer protocol. Thus, ZeroMQ is faster than Socket.IO [37,38]. However, ZeroMQ does not run in web browser. Moreover, ZeroMQ does not use 80 port or http port unlike Socket.IO so that any global-based networking system utilizing ZeroMQ should handle firewall problem. Sovcket.IO is seamless under MEARN stack environments. That is one of main reasons why the proposed MR-DSS chooses to base Socket.IO for providing real-time messaging service. Here, we provide testing results of the performance of real-time messaging service of the proposed MR-DSS, but we don't provide comparison between real-time messaging service of MR-DSS and Xibo's XMR since it is clear that Xibo's XMR is faster than the real-time messaging of the proposed MR-DSS [38,39].

For the performance evaluation, we utilized 'Artillery' tool [40] to test the networking performance of the Socket.IO handling module of the proposed MR-DSCMS. In 'Artillery' tool, testing environments are configured as follows. 100 users from our lab's computer client are set to continuously send the Socket.IO 'event' and 'status' message to the proposed MR-DSCMS during 30 seconds, which works at Google cloud platform under the same computing environments as that of the testing environments of REST APIs. Socket.IO 'event' message handling is by default supported by Socket.IO library and Socket.IO 'status' message handling is implemented in real-time messaging service module of MR-DSCMS, which returns success or failure response depending on the result of updating the status information into the MongoDB.

Table 3 shows testing results about performance evaluation of real-time communication service built on the top of Socket.IO in MR-DSS. In Table 3, '# of virtual users' means the number of virtual users created during 30 seconds, '#of Socket.IO messages sent' means total number of Socket.IO messages sent during 30 seconds, and 'RPS(Requests/sec)' is the average number of requests per second completed in the preceding 10 seconds (or throughout the test).

Experimental data in Table 3 with comparison to those in Table 1 and 2 show that Socket.IO-based real-time messaging is much faster than REST API-based web service. As it is now well known, in web application environments, Socket.IO is popular adopted in implementation of real-time messaging service since it is simple but fast enough.

5. CONCLUSIONS

In this paper, we proposed a MEARN stack-based real-time digital signage system (MR-DSS) which supports real-time operation for system management and monitoring, fast content update, instant messaging, as well as non real-time CMS services via REST APIs. In addition to explanation of component architecture, some design and implementation issues were clarified. Experimental results show that REST APIS of the proposed MR-DSS are efficiently implemented so that they are faster than those of an open source digital signage system, Xibo.

Currently, we are testing stability and scalability of MR-DSS under various environments. Scalability will be important for MR-DSSS since it aims

Table 3. Testing results of Real-time Communication sercives by Socket.IO messages in MR-DSS

Socket.IO message type	Success Rate (%)	Min Resp (ms)	Max Resp (ms)	Media Resp (ms)	99p Resp (ms)	RPS (reqs/sec)
status	100	0.2	11.3	0.4	2.3	49.16
event	100	0.1	3.1	0.3	1	49.2

to serve cloud-based worldwide digital signage service in the future. In addition to multiple Node.js application instances and MongoDB sharding, real-time communication service based on Socket.IO will be investigated experimentally about load performance, and the experimental results will be analyzed for improvement of scalability.

REFERENCES

[1] ITU-T Technology Watch Report, *Digital Signage: The Right Information in All the Right Places*, 2011.

[2] R. Gushue, What is a Digital Signage Content Management System?, https://enplug.com/blog/what-is-a-digital-signage-content-management-system. (accessed April., 25, 2017).

[3] Xibo,http://xibo.org.uk/. (accessed April., 25, 2017).

[4] Signagelive, https://signagelive.com/ (accessed April., 25, 2017).

[5] RESTful API, Learn RESTful: A RESTful Tutorial, http://www.RESTfulapitutorial.com/ (accessed April., 25, 2017).

[6] MEAN Stack, http://mean.io/ (accessed April., 25, 2017).

[7] MERN Stack, http://mern.io/ (accessed April., 25, 2017).

[8] MongoDB, https://www.mongodb.com/ (accessed April., 25, 2017).

[9] Express.js, https://expressjs.com/ (accessed April., 25, 2017).

[10] Node.js, https://nodejs.org/en/ (accessed April., 25, 2017).

[11] AngularJS, https://angularjs.org/ (accessed April., 25, 2017).

[12] ReactJS, https://facebook.github.io/react/ (accessed April., 25, 2017).

[13] Socket.IO, http://Socket.IO/ (accessed April., 25, 2017).

[14] JSON, http://json.org/ (accessed April., 25, 2017).

[15] Ghost, https://ghost.org/ (accessed April., 25, 2017).

[16] Keystone, http://keystonejs.com/ (accessed April., 25, 2017).

[17] Ulbora CMS, http://www.ulboracms.org/#!/ (accessed April., 25, 2017).

[18] Nginx, https://www.nginx.com/ (accessed April., 25, 2017).

[19] F. Cazenave, V. Quint, and C. Roisin, "Timesheets.js: When SMIL Meets HTML5 and CSS3," *Proceedings of the Eleventh ACM Symposium on Document Engineering, pp. 43-52, 2011.*

[20] W3C, SMIL 3.0 Timing and Synchronization, https://www.w3.org/TR/SMIL3/smil-timing.html (accessed April., 25, 2017).

[21] W3C, SMIL Timesheets 1.0, https://www.w3.org/TR/timesheets/. (accessed April., 25, 2017).

[22] Kyeong Hur, "Distributed Medium Access Control for N-Screen Multicast Services in Home Networks," *Journal of Korea Multimedia Society,* Vol. 19, No. 3, pp. 567-572, March 2016.

[23] POPAI, http://www.popia.com (accessed April., 25, 2017).

[24] Intel Digital Signage Technology, http://www.intel.com/content/www/us/en/retail/retail-digital-signage.html. (accessed April., 25, 2017).

[25] Digital Signage Federation, http://www.digitalsignagefederation.org/ (accessed April., 25, 2017).

[26] ITU-T H.780, *Digital Signage: Service Requirements and IPTV-based Architecture*, ITU-T SG16, 2012.

[27] W3C, Web Authentication Working Group Charter, https://www.w3.org/2015/12/web-authentication-charter.html (accessed April., 25, 2017).

[28] W3C, Web-based Signage Use Cases and Requirements, Final Business Group Report 21, https://www.w3.org/2016/websigns/ucr/

(accessed April., 25, 2017).

[29] W3C, Synchronized Multimedia Integration Language (SMIL 3.0), https://www.w3.org/TR/2008/REC-SMIL3-20081201/ (accessed April., 25, 2017).

[30] SCXML, State Chart XML (SCXML): State Machine Notation for Control Abstraction, https://www.w3.org/TR/scxml/ (accessed April., 25, 2017).

[31] WebRTC, https://webrtc.org/ (accessed April., 25, 2017).

[32] Electron, http://electron.atom.io/ (accessed April., 25, 2017).

[33] OAuth 2.0, https://oauth.net/2/ (accessed April., 25, 2017).

[34] S. Mumbaikar and P. Padiya, "Web Services Based On SOAP and RESTful Principles," *International Journal of Scientific and Research Publications*, Vol. 3, Issue 5, pp. 1-4, May 2013.

[35] Xibo-DSS APIs, http://xibo.org.uk/manual-tempel/api/ (accessed April., 25, 2017).

[36] Jmeter, http://jmeter.apache.org/usermanual/glossary.html (accessed April., 25, 2017).

[37] ZeroMQ, http://zeromq.org/ (accessed April., 25, 2017).

[38] Quora, How does Socket.IO Compare with a Message Queueing Service like ZeroMQ for Inter-process Communication?, https://www.quora.com/How-does-Socket-io-compare-with-a-message-queueing-service-like-ZeroMQ-for-inter-process-communication (accessed April., 25, 2017).

[39] InfiniBand Tests, http://zeromq.org/results:ib- tests-v206 (accessed April., 25, 2017).

[40] Artillery Tool, https://artillery.io/docs/ gettingstarted.html (accessed April., 25, 2017).

Khue Trinh Duy

He received the Engineer degree in computer engineering from the Hanoi University of Science and Technology, Hanoi, Vietnam, in 2014. He is currently a research assistant at Embedded Real-time Computing Laboratory, Soongsil University, Seoul, South Korea. His main areas of research interest are embedded systems, image processing, IO, mobile and web technology.

Thanh Binh Nguyen

He received the B. Eng. degree in computer science from the University of Science, Ho Chi Minh, Vietnam, in 2005, the M. Sc. degree in information and telecommunication engineering from the University of SoongSil, Seoul, South Korea, in 2010, and the Ph.D. degree in engineering at the University of SoongSil, Seoul, South Korea, in 2017. He is currently a principal software R&D researcher at Embedded Vision Inc., Seoul, South Korea, assistant at Embedded Real- time Computing Lab, University of Soongsil, Seoul, South Korea. His research interests cover the design and analysis of various smart embedded software system, I.O.T and also intelligent image, video analytic algorithms which is applied to visual surveillance, recognition systems, and etc.

Ukjin Jang

He received the Engineer degree in computer engineering from the Soongsil University, Seoul, Korea, in 2016. He is currently a research assistant at Embedded Real-time Computing Laboratory, Soongsil University, Seoul, South Korea. His main areas of research interest include modern web technology, and IoT.

ChanBin Kim

He received the B. Eng. degree in computer science from Soongsil University in 2014. He is currently a research assitant at Embedded Real-time Computing Laboratory, Soongsil University. His research interests include modern web technology, data mining system, and embedded systems.

Sun-Tae Chung

He received B.E. degree from Seoul National University, and M.S. degree and Ph.D. degree in Electrical Eng. and Computer Science from the University of Michigan, Ann Arbor, USA, in 1986 and 1990, respectively. Since 1991, he had been with the School of Electronic Eng. at the Soongsil university, Seoul, Korea where he is now a full professor. Now, he has been with the Dept. of Smart Systems Software, at the Soongsil Univ. since 2015. His research interests include digital signage, computer vision, visual surveillance, and embedded systems.