Sensors and Materials, Vol. 31, No. 10 (2019) 3229–3244
MYU Tokyo

S & M 2007

Dynamic Mashup Performance Comparison Using Open Application Programming Interface of Geoweb Map Platforms Available in Korea

Min-Soo Kim[1*] and In-Sung Jang[2]

[1]Department of Computer Engineering, Daejeon University,
62 Daehak-ro, Dong-gu, Daejeon 34520, Korea
[2]Hyper-connected Communication Research Laboratory, ETRI,
218 Gajeong-ro, Yuseong-gu, Daejeon 34129, Korea

(Received March 6, 2019; accepted April 30, 2019)

Keywords: mashup, Open API, map platform, geoweb, tile map

Recently, users have been more interested in the development of a dynamic mashup application that integrates and renders large amounts of real-time sensor data and geospatial data. However, such a dynamic mashup application has caused performance issues in large amounts of real-time sensor data integration and time series rendering. In this study, we compare and analyze the dynamic mashup performance characteristics of various types of map platform. For accurate performance comparison, we define performance comparison metrics of data loading time, mashup time, and user interaction time. We also implement a mashup prototype system composed of a Geoweb client, a sensor data server, and several map platforms in order to efficiently perform various experiments. The experimental results show the mashup performance characteristics of Google Maps, OpenStreetMap (OSM), VWorld, olleh Map, Daum Map, and Naver Map. Finally, we present which map platforms have good and stable performance in the dynamic mashup of large amounts of sensor data.

1. Introduction

With the advent of Geoweb 2.0 technologies, particularly Ajax, Open application programming interface (API), and tiling, various types of geospatial web application have been developed.(1) Particularly after the launch of Google Maps based on the Geoweb 2.0 technologies, the number of geospatial web applications that can mashup Google Maps with information from other web sites has greatly increased. Compared with the previous geospatial web applications that require significant programming skills, the new applications based on Open API allow users to easily share and integrate geospatial data and other information on the web.(2) Recently, users have been more interested in the development of geospatial mashup applications with the advent of many map platforms. For example, the map platforms of Google Maps,(3) Bing Maps,(4) and OpenStreetMap (OSM)(5) have been released. Such map platforms' front end normally utilizes Ajax technologies, and their back end offers Open API that allows

*Corresponding author: e-mail: minsoo@dju.ac.kr
https://doi.org/10.18494/SAM.2019.2365

ISSN 0914-4935

tiled maps to be embedded on third party web applications. In Korea, in addition to Google Maps, Bing Maps, and OSM, various map platforms such as Vworld,[6] olleh Map,[7] Daum Map,[8] and Naver Map[9] can be used in the mashup applications. Furthermore, recent mashup applications can integrate various types of geospatial data such as vector map, imagery map, street level map, 3D map, and indoor map. In particular, with the recent development of O2O, LBS, and Smart City services, the number of mashup applications integrating large amounts of third-party real-time sensor data has been increasing rapidly.[10,11] For example, third-party developers can embed real-time locations of moving objects and real-time weather, air quality, and traffic information into maps to create new dynamic mashup applications. However, such a dynamic mashup has caused performance issues in the front end owing to the large amounts of real-time sensor data integration and time series rendering.[12]

In this work, we try to compare and analyze the performance characteristics of various map platforms in the dynamic mashup that integrates large amounts of real-time sensor data. In the performance comparison using various map platforms, we compare the geospatial data loading time, the mashup time, and the time of user interaction such as zoom or pan. Specifically, we use map platforms of Google Maps, OSM, VWorld, olleh Map, Daum Map, and Naver Map, which are easily accessible in Korea. We also use various web browsers, namely, Chrome, Firefox, and Internet Explorer.

The remaining sections of this paper are structured as follows: we review related works regarding dynamic mashup applications using large amounts of real-time sensor data and their performance in Sect. 2. In Sect. 3, we propose the performance comparison methodology among various map platforms. In Sect. 4, we show a series of experiments on the performance analysis and discuss the results. In Sect. 5, we conclude this work.

2. Related Works

There have been several studies on the performance of Geoweb services. When the early GeoWeb services facilitated geospatial data sharing through the web, researchers first were interested in the performance improvement of Open GIS Consortium (OGC) Web Map Service (WMS).[13–18] Loechel and Schmid[14] argued that fast response time to a web request is a mandatory performance of WMS. They proposed several caching techniques such as tile caching and reverse proxy caching for the fast response of WMS. For example, the caching techniques had limits of 5 s maximum for a full map load and 2 s maximum for visual feedback for military situational awareness system application.[19] Schmid and Reinhardt[15] argued that Geoweb services require a certain quality of service (QoS) and the QoS becomes increasingly important. They presented a general evaluation procedure for the QoS and demonstrated its applicability using WMS. Specifically, they defined a goal as the fast response of geospatial data to quite a number of users and analyzed the response time using WMS.

Yang *et al.*[16] argued that WMS servers are not well optimized for real-world requirements and WMS clients are not well designed for multiple connections to WMS servers. Thus, they analyzed performance patterns of several WMS servers and proposed a new design of a WMS client by using the patterns. To improve the performance of a WMS server, Garcia *et al.*[18]

proposed algorithms to optimize WMS tile caching (WMS-C). The WMS-C approach divides the map into a discrete set of images, called tiles, and restricts user requests to the set of images.[20] The Open Source Geospatial Foundation (OSGeo) announced WMS-C, and OGC released the Web Map Tile Service (WMTS) Standard[21] inspired by the WMS-C. Wu *et al.*[17] argued that it is significant and challenging to select a satisfactory WMS server, because it requires the consideration of multiple factors such as user preference, quality of data, and performance. Thus, they proposed a web portal that can explore and compare the service quality of many WMS servers. The web portal allows a user to query and filter a WMS server using various types of user interaction. Schmid *et al.*[13] presented a WMS server using NoSQL as well as the SQL database for huge amounts of geospatial data storage. They also compared the performance characteristics of two WMS servers, which consisted of PostgreSQL and MongoDB.

Besides the performance improvement of OGC WMS, there have been several studies related to the performance of general Geoweb services.[22–25] Zhang[25] proposed a main-memory-based quad tree to efficiently support dynamic geospatial window queries. He used the quad tree to improve the performance of window queries on the Geoweb services. Huang and Chang[23] proposed a Geoweb search engine called Geoweb Crawler that can efficiently find various Geoweb resources. To improve the performance of the Geoweb Crawler, they used the MapReduce concept[26] to execute the crawling process in parallel. The Geoweb Crawler could efficiently find Geoweb resources such as OGC web services, KML, and shape data. Sani and Rinner[22] were interested in scalability, which was identified as a key concern with participatory Geoweb applications. They argued that scalability describes the ability of a Geoweb server to accommodate an increased number of users through the addition of system resources. Thus, they proposed a cloud computing implementation for the scalability and performance improvement of a Geoweb server. Dalton[24] analyzed the social and technological limits and possibilities of third-party Geoweb applications based on web services such as Google Maps.

In recent years, Geoweb 2.0 technology and standards have transformed a simple Geoweb server into a Geoweb platform. The Geoweb platform enables the creation of new knowledge and applications through the mashup of user data and geospatial data. Karnatak *et al.*[27] and Gong *et al.*[28] proposed geospatial mashup solutions for disaster management and urban management, respectively. They provided online access, visualization, geospatial analysis, and real-time sensor data sharing services following a mashup framework of the Geoweb 2.0. They aimed at contributing to sustainable disaster management and urban management by using a Geoweb 2.0 map platform to quickly and easily develop mashup solutions.

Thus far, these studies on the performance of Geoweb services have been mainly interested in the improvement of the Geoweb server or Geoweb search engine. They never took the performance of a Geoweb client into account. Therefore, we try to analyze the mashup performance of a web client that integrates large amounts of real-time sensor data with geospatial data using the Geoweb 2.0 technologies. The performance analysis will show how many real-time data can be efficiently mashed up on the Geoweb 2.0 map platforms.

3. Performance Comparison of Map Platforms in the Dynamic Mashup

3.1 Performance comparison metrics

We present a performance comparison metrics of various Geoweb 2.0 map platforms in the dynamic mashup that integrates large amounts of time series data with geospatial data. A Geoweb client performs such a dynamic mashup using Open API of Geoweb 2.0 platforms. In the dynamic mashup process, a Geoweb client performs several time-consuming tasks. The tasks are data loading, mashup, and user interaction as shown in Fig. 1. The data loading task downloads geospatial data from a map platform and loads user sensor data from a client's data store. The mashup task processes the real-time integration and time series rendering of geospatial data and user sensor data. The user interaction task processes the real-time rendering of the data integrated into a zoom or pan operation. The performance of three tasks depends mainly on the Open API performance of a map platform. Therefore, we try to compare the performance characteristics of the three tasks in relation to the Open API performance. In particular, we propose a performance comparison methodology to be able to accurately compare the total time spent in three main tasks.

As shown in Fig. 1, we perform three main tasks of data loading, mashup, and user interaction in this order. In this study, we define the total dynamic mashup performance of a map platform as a sum of each task performance, and each performance is measured by experiment on each task. The total dynamic mashup performance can be defined as

$$\begin{aligned} \textit{Total Performance } (T_p) = \textit{Data Loading Time } (T_d) + \textit{Mashup Time } (T_m) \\ + \textit{User Interaction Time } (T_i). \end{aligned} \tag{1}$$

First, T_d means the time it takes for a Geoweb client to load data. A Geoweb client may load geospatial data and user sensor data. However, we consider that the loading time of user sensor data is independent of the mashup performance of a map platform, since user data are directly

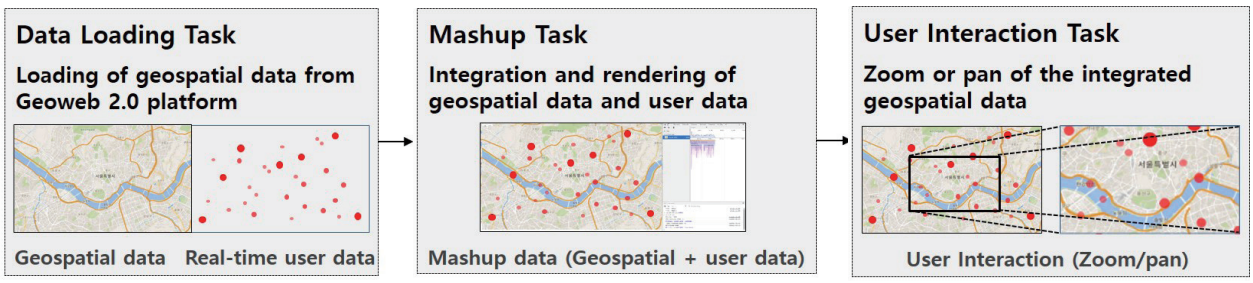


Fig. 1. (Color online) Three main tasks in a dynamic mashup: (a) data loading, (b) mashup, and (c) user interaction.

loaded from user's local data store. Therefore, we define T_d as the loading time of geospatial data only. Second, T_m means the time it takes for a Geoweb client to mashup the loaded data. The mashup task is composed of the real-time integration and time series rendering of the loaded data. Specifically, a Geoweb client integrates geospatial data with user sensor data to create an integrated layer and renders the integrated layer. Therefore, we define T_m as the total time for the integration and rendering. Third, T_i means the time it takes for a Geoweb client to process user interaction on the integrated layer. User interactions such as zoom or pan can often be used in the dynamic mashup. Moreover, they require new data loading and mashup tasks frequently. Therefore, we think that user interaction can be an important factor in measuring the performance of a map platform. We define T_i as the total time to perform such user interaction.

From T_d, we can see the initialization time of a Geoweb client. In the initialization, a Geoweb client downloads an Open API module and initial geospatial data from a map platform. Although T_d may vary depending on not only Open API but also a map platform status such as network condition, installation location, and hardware specifications, it has an important meaning in terms of the initialization performance in the dynamic mashup. From T_m, we can see the integration and rendering time consumed by Open API operations. In other words, T_m measures the performance of only Open API in a Geoweb client. Therefore, we can exactly compare the Open API performance characteristics of various types of map platform in the process of integration and rendering. From T_i, we can see the user interaction time consumed by Open API in a zoom or pan operation. The user interaction sometimes requires new data loading as well as integration and rendering. Finally, we think that T_d, T_m, and T_i are meaningful in the performance measurement in the dynamic mashup. Also, we think that T_p is needed to measure the total performance of the dynamic mashup, since normal mashup applications perform all of the initialization, integration and rendering, and user interaction. In Sect. 3.2, we will explain how to measure T_d, T_m, T_i, and T_p specifically.

3.2 Prototype system for performance measurement

In this subsection, we implement a mashup prototype system shown in Fig. 2, which can efficiently measure T_d, T_m, and T_i. The system consists of three parts: a Geoweb client, a user data server, and six types of Geoweb 2.0 map platform.

First, we implement a user data server in order to use it as an input source of large amounts of real-time sensor data. The server is implemented using a web server of Node.js and a database system of PostgreSQL. The server only provides a Geoweb client with real-time sensor data that would be integrated with geospatial data. We use the server as a client's data store for user sensor data. We exclude the loading time from the user data server in T_d, as mentioned in Sect. 3.1.

Second, we implement a Geoweb client using Open APIs of six types of Geoweb 2.0 map platform. The Geoweb client consists of three performance measurement modules for the main tasks of data loading, mashup, and user interaction. On the other hand, the Geoweb client based on JavaScript can be implemented using the web browsers Chrome, Firefox, and Internet

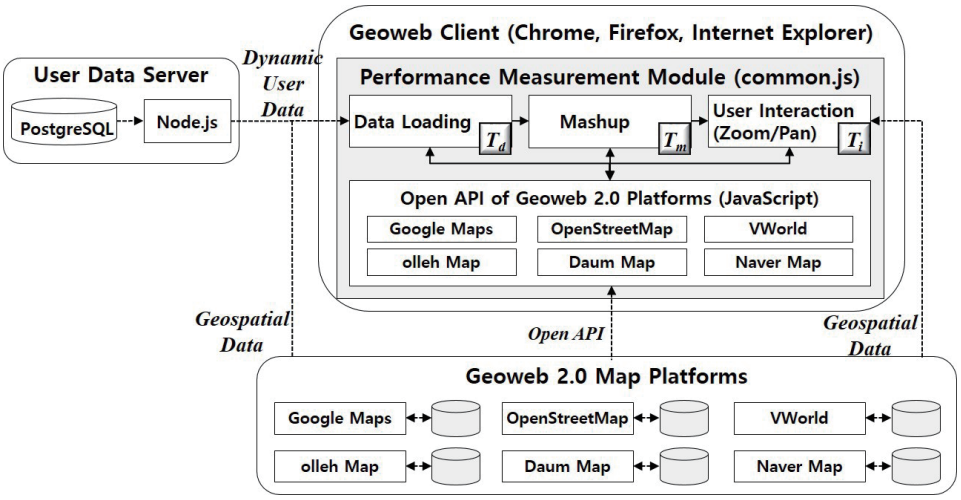


Fig. 2. Mashup prototype system composed of a Geoweb client, a user data server, and six types of Geoweb 2.0 map platform.

Explorer. In the data loading module, we measure T_d, which is the time spent in loading a discrete set of tiled images within the experimental map extent from a map platform. T_d is measured in two cases with and without a web browser cache specifically. Such a web browser cache can significantly improve T_d when a Geoweb client repeatedly loads tiled images within a similar map extent. In the mashup module, we measure T_m, the time spent in integrating and rendering geospatial data and user sensor data. T_m is basically measured in order to examine the change in mashup performance as the number of user sensor data increases. In this study, we vary the number of user sensor data from 1000 to 20000 to see enough performance difference in a Geoweb client. T_m is measured in two rendering cases of a vector layer and a heat map layer. T_{m_v} and T_{m_h} are measured on a vector layer and a heat map layer, respectively. In the user interaction module, we measure T_i, the time spent in processing user interaction such as zoom or pan. T_i is measured in order to examine the change in mashup and rendering performance when user interaction occurs in a Geoweb client. In this study, we process one level zoom in and pan as much as the current map size to see the performance change. T_i is also measured in two rendering cases, namely, T_{i_v} and T_{i_h}.

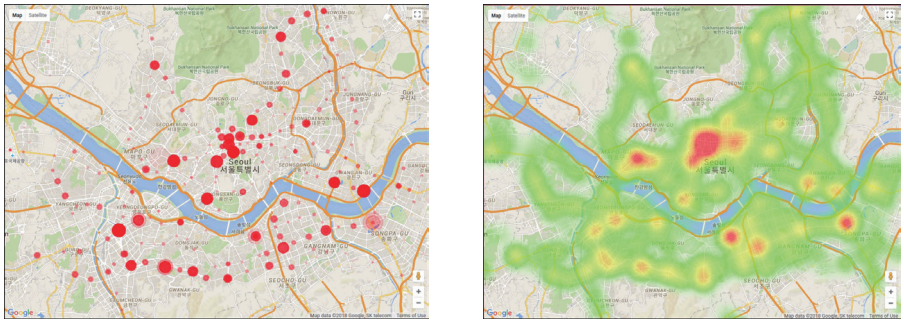
Third, we use six types of GeoWeb 2.0 map platform as an input source of geospatial data. As shown in Fig. 2, each map platform sends geospatial data to a Geoweb client upon the request of the data loading and user interaction modules. Each map platform also sends an Open API module to a Geoweb client. In this study, we try to compare the performance characteristics of six types of map platform to find out the best map platform for dynamic mashup application.

4. Performance Study

We run our experiments on map platforms of Google Maps, OSM, VWorld, olleh Map, Daum Map, and Naver Map, which are readily available in Korea. We also run experiments on

three web browsers, namely, Chrome, Firefox, and Internet Explorer. As a user sensor data set, we use the number of people getting on and off by time of all subway stations in Seoul. The user sensor data set is an actual data set collected from the data portal of the city of Seoul. As a geospatial data set, we use tiled images that are contained within the map extent of the entire area of Seoul. Finally, we run experiments on two rendering cases of a vector layer and a heat map layer, which are mostly used to render time-series data set in real time. Figure 3 shows examples of a vector layer and a heat map layer. The vector layer is rendered using circles whose area increases with respect to the increase in the number of people in all subway stations.

To obtain accurate experimental results, we run our experiments five times using the same experimental setups and calculate the average. Table 1 summarizes the experimental setups used in our experiments.



(a) (b)

Fig. 3. (Color online) Two rendering cases on the number of people by hour of day in all subway stations in Seoul: (a) rendering using a vector layer and (b) rendering using a heat map layer.

Table 1
Experimental setups of a map platform, a web browser, a rendering case, and a user sensor data set.

Category	Setups	Features
Map platform	- Google Maps 3.30	- World 2D,3D map, TLS[1](3)
	- OpenStreetMap 3.0	- World 2D map, TLS, OpenLayers compatibility(5)
	- VWorld 2.0	- Domestic 2D/3D map, OpenLayers compatibility(6)
	- olleh Map 3.0	- Domestic 2D map, OpenLayers compatibility(7)
	- Daum Map 3.0	- Domestic 2D map(8)
	- Naver Map 3.1.0	- Domestic 2D map(9)
Web browser	- Internet Explorer 11 - Chrome 66 - Firefox 60	- With and without a web browser cache
Rendering	- Vector layer - Heat map layer	- For time-series data, a vector layer using a circle whose area changes and a heat map layer
User sensor data set	- Time series data	- Time series data from 1000 to 20000 - Number of people by hour of day in all subway stations within the experimental map extent (Seoul)
Geospatial data set	- Tiled images	- Tiled images within the experimental map extent

[1]Transport Layer Security (TLS) is a cryptographic protocol that provides communications security over a network.

4.1 Experimental results of data loading time (T_d)

In this experiment, we measure the data loading time (T_d) of geospatial data. In other words, we measure the time it takes for a Geoweb client to load and render the tiled images of geospatial data. We perform the experiment with the most minimum load on the map platform as possible. Table 2 shows the experimental results of T_d for six types of map platform shown in Table 1.

First, with no browser cache, OSM needs too much T_d from 2939.2 to 6843.3 ms and Google Maps needs somewhat much T_d from 604.8 to 684.6 ms. Other map platforms need much less T_d than OSM and Google Maps. T_d with no browser cache depends on the server-side performance of a map platform. The server-side performance is typically determined by the throughput and response time of a server system. The throughput and response time of OSM are not comparable to those of other commercial map platforms since OSM is a free world map platform located overseas. Therefore, we think that the experimental result for OSM is understandable. In contrast, even though Google Maps is also located overseas, it shows relatively good performance to be used in Korea. We think that this is because of the powerful throughput of the Google Cloud platform all over the world and the outstanding response time of Google Maps. Other domestic map platforms of VWorld, olleh Map, Naver Map, and Daum Map show better performance than OSM and Google Maps. That is because the map platforms are in Korea and they have to provide commercial services with high performance. T_d is measured only from 227.2 to 448.4 ms at the maximum in Table 2.

Second, with a browser cache, all map platforms need T_d of less than only 244.4 ms on all web browsers. That is because a Geoweb client may load geospatial data from a browser cache rather than a map platform and most T_d has only rendering time with little loading time. Google Maps unusually increases T_d slightly. We think that Google Maps does additional work for some advanced rendering. However, such a slight increase does not make a big difference in T_d.

Third, we calculate each difference in order to see how much time difference in T_d exists among web browsers. We calculate the difference except for OSM, which needs unusually too much T_d. The calculation results are

$$-14.2 \text{ ms} \leq T_d \textit{ in Chrome} - T_d \textit{ in Firefox} \leq -79.8 \text{ ms}, \tag{2}$$

$$-4.4 \text{ ms} \leq T_d \textit{ in Chrome} - T_d \textit{ in Internet Explorer} \leq -23.8 \text{ ms}, \tag{3}$$

Table 2
Experimental results of the data loading time in milliseconds for six types of map platform.

Web browser		Google	OSM	VWorld	olleh	Naver	Daum
Chrome	No Cache	604.8	4275.4	227.2	448.4	307.8	238.4
	Cache	192.0	46.8	88.8	21.6	37.0	11.8
Firefox	No Cache	684.6	6843.4	277.6	473.0	383.8	253.8
	Cache	223.6	65.2	103.0	45.4	115.6	34.6
Internet Explorer	No Cache	607.6	2939.2	231.6	441.6	302.0	232.0
	Cache	244.4	52.2	65.0	32.4	70.6	16.0

$$-20.8 \text{ ms} \leq T_d \textit{ in Firefox} - T_d \textit{ in Internet Explorer} \leq 81.8 \text{ ms}. \quad (4)$$

From Eqs. (2)–(4), it can be seen that there is no big difference in T_d among web browsers. There is a small time difference from the minimum 0.0044 s to the maximum 0.0818 s. It can be said that a web browser does not have a markedly large effect on T_d.

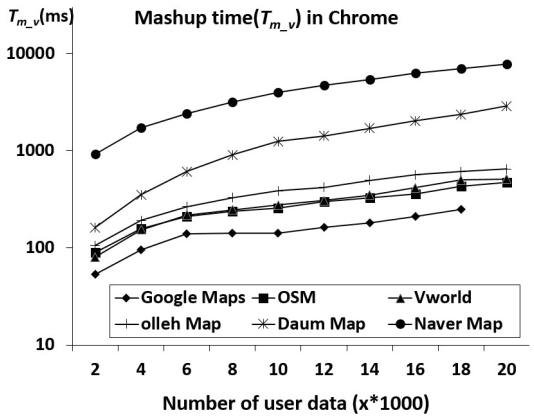
In conclusion, the performance of T_d seems to have no problem building mashup applications using all map platforms except for OSM having no cache. Even for OSM, although it takes much time for the initial loading of geospatial data, it seems that there is no problem to build mashup applications after the initial loading.

4.2 Experimental results of mashup time (T_m)

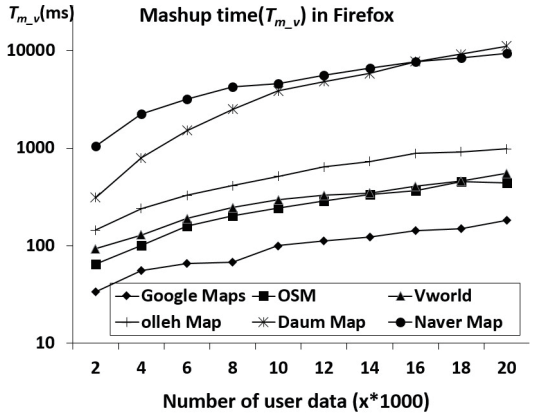
In this experiment, we measure the mashup time (T_m) of geospatial data and user data. In other words, we measure the time at which Open API is called for the mashup and then the rendering of the mashup data is complete. We perform the same experiment on the vector and heat map layers to obtain the results of T_{m_v} and T_{m_h}. Figure 4 shows T_{m_v} and T_{m_h} for map platforms shown in Table 1. The *X*-axis represents 2000 to 20000 user data and the *Y*-axis represents the mashup time in milliseconds. In particular, a base-10 log scale is used for the *Y*-axis for a large range of quantities of T_{m_v} and T_{m_h}.

First, as shown in Figs. 4(a)–4(c), Google Maps, OSM, VWorld, and olleh Map show good performance at T_{m_v} of less than one second for the maximum number of user data. On the other hand, Naver Map and Daum Map show poor performance at T_{m_v} of 10 s or more for the maximum number of user data. Naver Map and Daum map need T_{m_v} of 7822.2 to 22866.8 ms and 2874 to 22780.8 ms, respectively, for 20000 user data in three web browsers. Therefore, we can see that Naver Map and Daum Map may have difficulty in the mashup of real applications, because T_{m_v} increases greatly when the number of user data increases. In addition, we can see that there is a difference in the T_{m_v} of each map platform for each web browser and that the difference increases much more as the number of user data increases. In Figs. 4(a)–4(c), Daum Map and Naver Map show that Internet Explorer generally requires much more T_{m_v} than Chrome and Firefox. For example, the T_{m_v} of Naver Map increases up to 2.9 times from 7822.2 to 22866.8 ms and the T_{m_v} of Daum Map increases up to 7.9 times from 2874 to 22780.8 ms. Therefore, we can see that a mashup application with a large number of user data can benefit from using Chrome or Firefox rather than Internet Explorer.

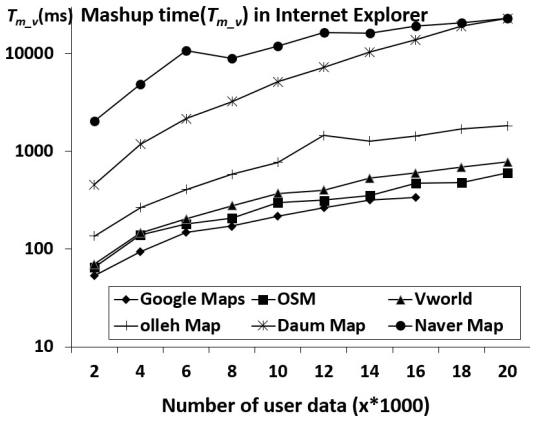
Second, as shown in Figs. 4(d)–4(f), we obtain T_{m_h} for the heat map layer of only four map platforms. Daum Map and olleh Map do not support Open API for the heat map layer. Figures 4(d)–4(f) show that T_{m_h} has a similar pattern to the previous T_{m_v} in Figs. 4(a)–4(c). Google Maps, OSM, and VWorld still show good performance at T_{m_h} of less than one second for the maximum number of user data. Naver Map shows the worst performance at T_{m_h} of 2.4 to 21.7 s in Firefox. For each web browser, Google Maps, OSM, and VWorld have almost no difference at T_{m_h} of less than 0.2 s, but Naver Map has a large difference at T_{m_h} from 1.9 s in Chrome and 12.5 s in Internet Explorer to even 21.7 s in Firefox. Therefore, we can also see that Naver Map may have difficulty in mashup as shown in Figs. 4(d)–4(f) when the number of user data increases even when using a heat map.



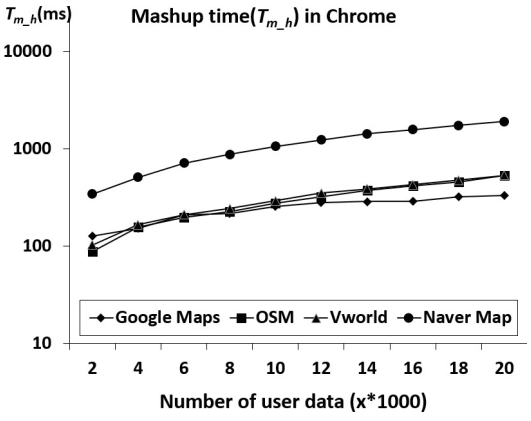
(a)



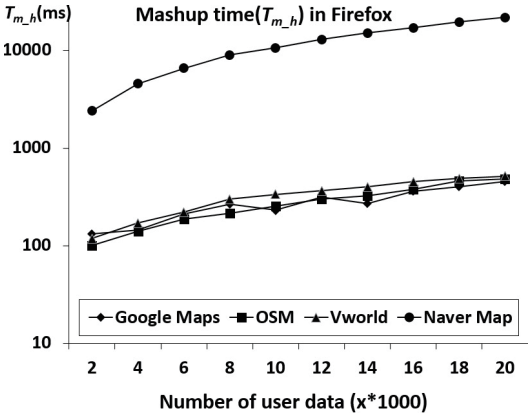
(b)



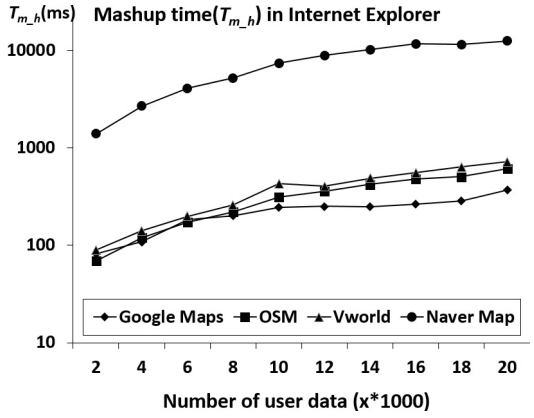
(c)



(d)



(e)



(f)

Fig. 4. Experimental results of the mashup time in milliseconds for six types of map platform: (a)–(c) result (T_{m_v}) for the vector layer and (d)–(f) result (T_{m_h}) for the heat map layer.

Third, except for Naver Map, there is no significant difference between T_{m_v} and T_{m_h} in the two rendering cases. In Fig. 4, we can see a small difference of less than 0.27 s for Google Map and a difference of less than 0.06 s for OSM and VWorld. In contrast, Naver Map has a large difference. Naver Map renders the heat map faster than the vector map by 5.9 and 10.3 s in Chrome and Internet Explorer, respectively, and renders the vector map faster than the heat map by 12.3 s in Firefox.

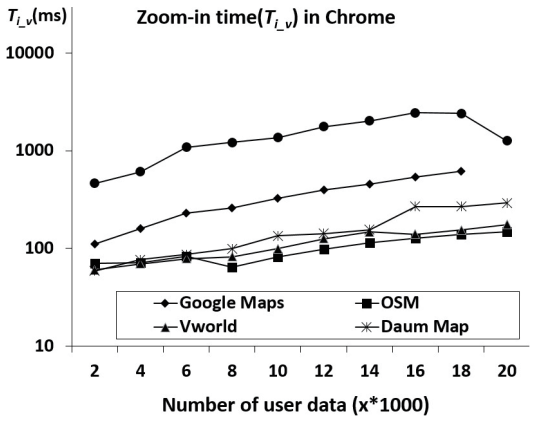
Finally, we can conclude that a user can build a mashup application using any map platform regardless of the result of T_m except for Naver Map and Daum Map when the number of user data increases significantly.

4.3 Experimental results of user interaction time (T_i)

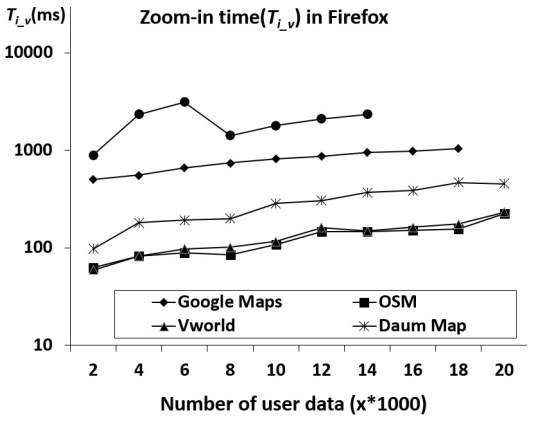
In this subsection, we discuss the results of the experiment with only five or four map platforms excluding olleh Map or Daum Map. The experiment excludes olleh Map that does not support Open API for zoom-in and panning in Figs. 5(a)–5(c) and 6(a)–6(c). It also excludes olleh Map and Daum Map that does not support Open API for a heat map in Figs. 5(d)–5(f) and 6(d)–6(f). We measure the user interaction time (T_i) at which Open API is called for zoom and pan and then their renderings are complete. We perform this experiment on the vector and heat map layers as described in Sect. 4.2. Figures 5 and 6 show T_{i_v} and T_{i_h} for zoom and pan, respectively. The *X*-axis represents the number of user data and the *Y*-axis represents the user interaction time in milliseconds of a base-10 log scale for T_{i_v} and T_{i_h}, as shown in Fig. 4.

As shown in Figs. 5(a)–5(c), OSM and VWorld show the best performance, followed by Daum Map, Google Map, and Naver Map. OSM and VWorld show T_{i_v} of 149.2 to 310.6 ms and 175.8 to 421.6 ms, respectively, for 20000 user data in three web browsers. Daum Map and Google Maps show a slightly higher T_{i_v} than OSM and VWorld. For example, Daum Map shows T_{i_v} values of 292.9, 231.2, and 1306 ms for 20000 user data in Chrome, Firefox, and Internet Explorer, respectively. Finally, Naver Map shows the worst performance at T_{i_v} of 3819.8 ms for 20000 user data in Fig. 5(c). Unlike Figs. 5(a)–5(c), we can see a performance result slightly different from that shown in Figs. 5(d)–5(f). All map platforms except for Naver Map show good performance at T_{i_h} of less than 231 ms in the worst case. Only Naver Map shows very poor performance at T_{i_h} of 1334.2 to 8597.8 ms. This experiment shows that the zoom interaction works well on all map platforms except Naver Map, regardless of the number of user data and type of web browser.

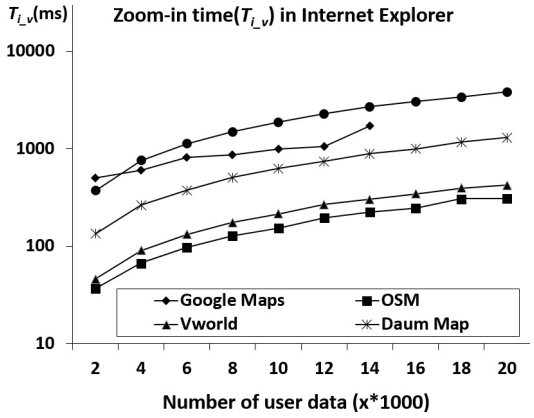
Figure 6 shows a performance pattern similar to that shown in Fig. 5. OSM and VWorld show the best performance, whereas Daum Map and Google Maps show mid-range performance among all map platforms. Naver Map shows the worst performance. On the other hand, we can see that the panning interaction is performed slightly faster than the zoom interaction in most cases from Figs. 5 and 6. For example, in almost all cases except for Google Maps in Figs. 6(d)–6(f), we can see that T_{i_v} and T_{i_h} in Fig. 6 are less than those in Fig. 5 when the number of user data increases. This experiment also shows that the panning interaction works well on all map platforms except Naver Map, regardless of the number of user data and type of web browser.



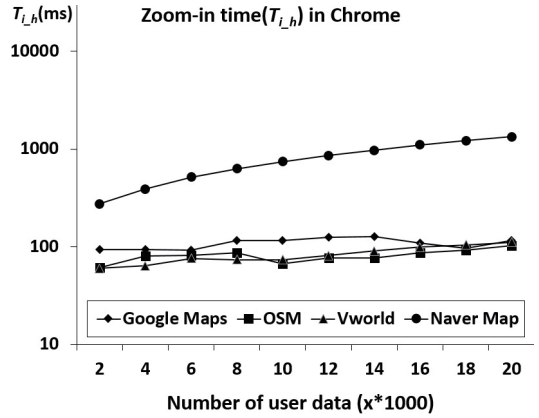
(a)



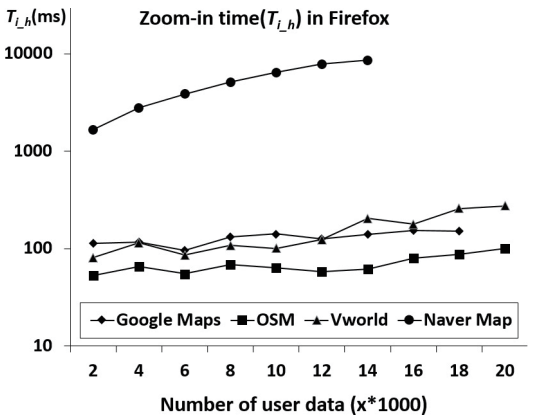
(b)



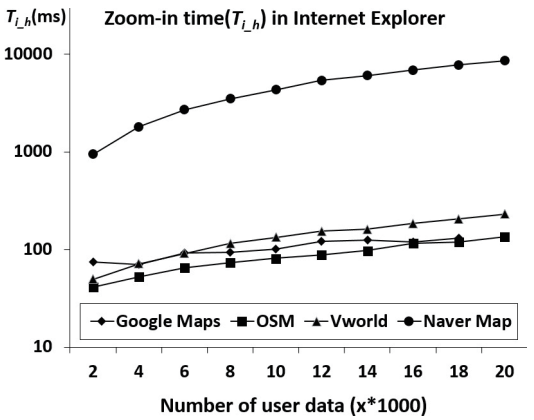
(c)



(d)

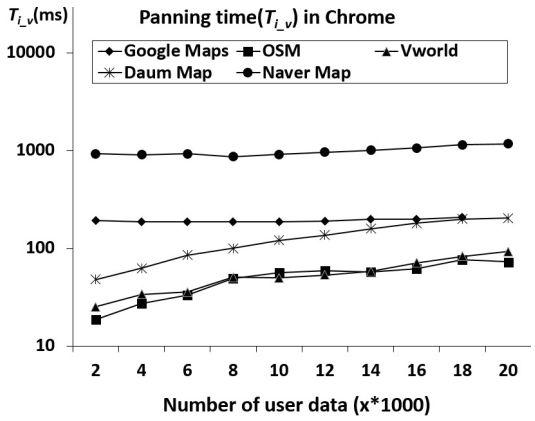


(e)

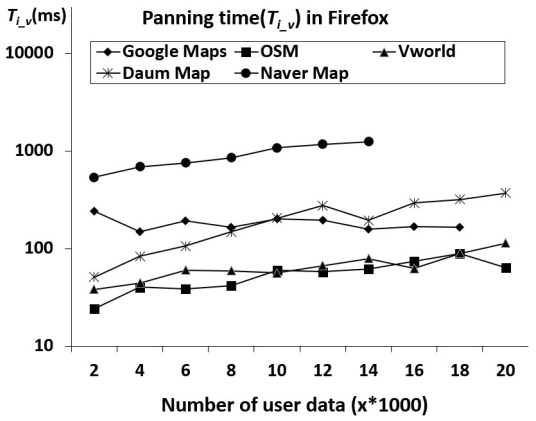


(f)

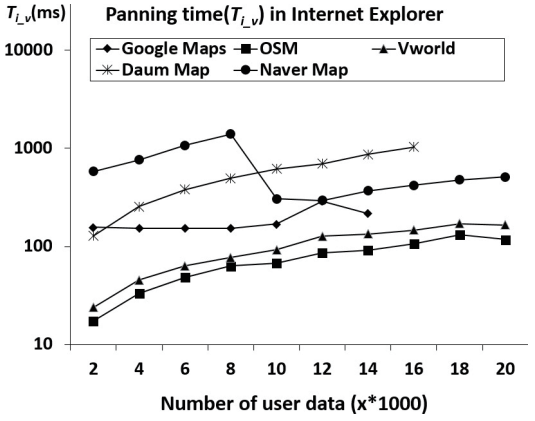
Fig. 5. Experimental results of zoom time in milliseconds for map platforms: (a)–(c) result (T_{i_v}) for the vector layer and (d)–(f) result (T_{i_h}) for the heat map layer.



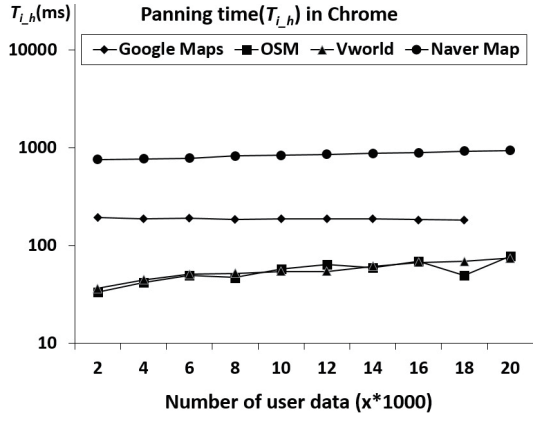
(a)



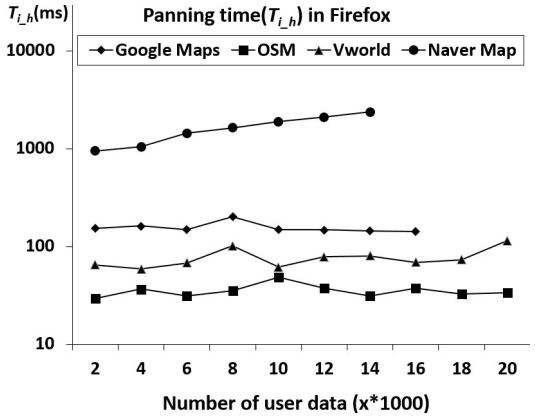
(b)



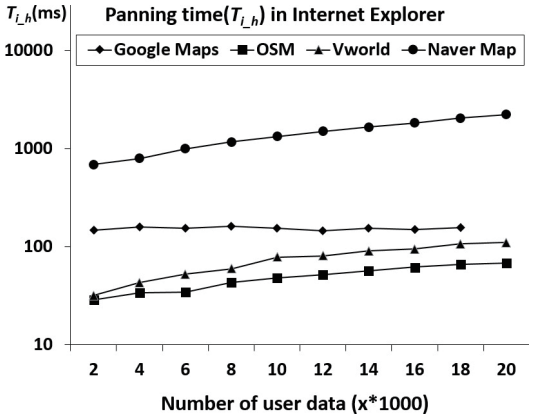
(c)



(d)



(e)



(f)

Fig. 6. Experimental results of panning time in milliseconds for map platforms: (a)–(c) result (T_{i_v}) for the vector layer and (d)–(f) result (T_{i_h}) for the heat map layer.

Finally, we can conclude that the user can perform user interaction with good performance in most cases except for Naver Map when the number of user data increases significantly.

4.4 Discussion

Here, we summarize the experimental results of the data loading, mashup, and user interaction time. Table 3 shows the total mashup performance (T_p) consisting of the sum of T_d, T_m, and T_i, which was defined in Sect. 3.1. In Table 3, we calculate T_p using the T_d, T_m, and T_i results when the number of user data is the median of 10000. In more detail, in order to calculate T_p, we use T_d when the web browser cache is applied, use each T_{m_v} and T_{m_h} considering the vector and heat map layers, and also use each average of T_{i_v} and T_{i_h} for zoom and pan.

We define T_p as a simple sum of T_d, T_m, and T_i in order to compare the total time that each map platform actually consumes in a mashup application. First, olleh Map and Daum Map do not provide Open API for the heat map or user interaction, so it is difficult to compare their T_p values with those of the other map platforms. However, we can easily guess from the lack of such an Open API that a user's preference for the two map platforms is worse than the others.

Second, the comparison of T_p by each web browser shows a good result in the order of Chrome, Firefox, and Internet Explorer. For example, except for olleh Map and Daum Map, Chrome shows 711.7 to 7020.6 ms, Firefox 702 to 20961.2 ms, and Internet Explorer 838.6 to 23337.7 ms. In Google Maps, OSM, and VWorld, the maximum differences in T_p among web browsers are about 0.41, 0.14, and 0.33 s, respectively, so we can see that there is not much difference when using any web browser. However, the maximum difference in the T_p of Naver Map is about 16.3 s; thus, we can see that it is certainly advantageous to use Chrome.

Table 3
Comparison of total performance (T_p) values among Chrome, Firefox, and Internet Explorer.

Browser	Performance	Google	OSM	VWorld	olleh	Daum	Naver
Chrome	T_d	192.0	46.8	88.8	21.6	11.8	37.0
	T_{m_v}	142.2	255.6	275.6	385.4	1241.2	3989.6
	T_{m_h}	256.2	277.4	291.2	—	—	1063.2
	T_{i_v}	256.8	69.5	74.6	—	127.9	1137.8
	T_{i_h}	152.6	62.4	64.1	—	—	793.0
	T_p	999.8	711.7	794.3	407	1380.9	7020.6
Firefox	T_d	223.6	65.2	103.0	45.4	34.6	115.6
	T_{m_v}	99.8	242.0	295.4	514.6	3887.6	4623.6
	T_{m_h}	230.0	254.6	335.4	—	—	10573.2
	T_{i_v}	505.6	83.9	86.1	—	245.2	1438.8
	T_{i_h}	145.3	56.3	81.2	—	—	4165.0
	T_p	1204.3	702.0	901.1	560.0	4167.4	20916.2
Internet Explorer	T_d	244.4	52.2	65.0	32.4	16.0	70.6
	T_{m_v}	217.2	299.2	371.2	768.0	5115.0	11884.6
	T_{m_h}	245.4	312.0	429.6	—	—	7451.6
	T_{i_v}	579.6	110.7	153.9	—	623.6	1093.7
	T_{i_h}	127.4	64.5	105.9	—	—	2837.2
	T_p	1414	838.6	1125.6	800.4	5754.6	23337.7

Third, the comparison of T_p by each map platform shows a good result in the order of OSM, VWorld, Google Maps, and Naver Map. For example, OSM shows 702 to 838.6 ms, VWorld 794.3 to 20961.2 ms, Google Maps 999.8 to 1414 ms, and Naver Map 7020.6 to 23337.7 ms. Except for Naver Map, the maximum differences in T_p among map platforms are 0.29 s for Chrome, 0.5 s for Firefox, and 0.58 s for Internet Explorer, so there is little difference when using any map platform. However, the maximum differences of Naver Map are about 6.3 s for Chrome, 20.2 s for Firefox, and 22.5 s for Internet Explorer, so we can see that it is certainly advantageous to use other map platforms.

Most users think that all map platforms may support Open API for time-efficient mashup and user interaction. However, we can see that the T_p values of only Google Maps, OSM, and VWorld show good performance despite the increase in the number of user data from our experiments. In addition, we can see that Google Maps sometimes cannot perform the mashup and user interaction when the number of user data increases to nearly 20000, and Daum Map and olleh Map do not provide Open API for the heat map and user interaction. Therefore, from only the T_p perspective, we can conclude that OSM and VWorld are the most desirable for mashup applications that use a lot of dynamic user data. Besides T_p, there may be the type, quality, and service range of geospatial data and the degree of free use of Open API as the important criteria for a user to select a map platform. However, we focus on only T_p and exclude the above criteria from this discussion.

5. Conclusions

In this study, we compared and analyzed the dynamic mashup performance characteristics of various map platforms. To efficiently compare the performance, we defined and measured performance comparison metrics of the data loading time (T_d), mashup time (T_m), and user interaction time (T_i). We also implemented a mashup prototype system composed of a Geoweb client, a user data server, and map platforms to accurately measure the performance.

Our experimental results showed that Google Maps, OSM, and VWorld generally outperform the others in T_p. However, Google Maps sometimes could not perform a time-series mashup when the number of user sensor data increases to nearly 20000. As a result, any map platform can be used if the number of user sensor data is very small or it is not a time series mashup. However, when performing a time series mashup for large amounts of sensor data, we think that it is better to use OSM and VWorld, if possible.

We plan to extend this work in two directions. First, we would like to define new performance comparison metrics on a dynamic mashup. We expect that new performance metrics can more accurately measure the mashup performance of map platforms. Second, we would like to propose a map platform selection criterion for users considering various situations such as the quality and service range of geospatial data and the degree of free use of Open API. We expect it to be a more realistic selection criterion.

Acknowledgments

This research was supported by a grant#(19DRMS-B147287-02) from the development of customized realistic 3D geospatial information update and utilization technology based on consumer demand, funded by the Ministry of Land, Infrastructure and Transport of Korean government.

References

1 S. Gabriel: Beginning Google Maps API3 (Apress, New York, 2010) p. 1. https://doi.org/10.1007/978-1-4302-2803-5
2 S. Jun and S. Lee: Spatial Inf. Res. **25** (2017) 725. https://doi.org/10.1007/s41324-017-0138-y
3 Google Maps: https://www.google.com/maps (accessed March 2019).
4 Bing Maps: https://www.bing.com/maps (accessed March 2019).
5 OpenStreetMap: https://www.openstreetmap.org/ (accessed March 2019).
6 VWorld: http://map.vworld.kr/map/ (accessed March 2019).
7 olleh Map: https://www.apistore.co.kr/api/apiList.do (accessed March 2019).
8 Daum Map: https://map.kakao.com/ (accessed March 2019).
9 Naver Map: https://map.naver.com/ (accessed March 2019).
10 M. Kim: Spatial Inf. Res. **26** (2018) 113. https://doi.org/10.1007/s41324-017-0161-z
11 M. Kim: Spatial Inf. Res. **25** (2017) 735. https://doi.org/10.1007/s41324-017-0131-5
12 H. Jang, D. Kim, J. Kim, and I. Jang: Spatial Inf. Res. **24** (2016) 367. https://doi.org/10.1007/s41324-016-0038-6
13 S. Schmid, E. Galicz, and W. Reinhardt: Proc. 2015 Int. Conf. Military Technologies (ICMT 2015) 1. https://doi.org/10.1109/MILTECHS.2015.7153736
14 A. J. Loechel and S. Schmid: IJSDIR. **8** (2013) 43. https://doi.org/10.2902/1725-0463.2013.08.art3
15 S. Schmid and W. Reinhardt: Proc. 2015 Int. Cartographic Conf. (ICC 2015) 23.
16 P. Yang, Y. Cao, and J. Evans: GIScience Remote Sens. **44** (2007) 320. https://doi.org/10.2747/1548-1603.44.4.320
17 S. Wu, M. Zhang, Q. Huang, Y. Zhang, C. Wan, K. Zhang, J. Cao, Z. Gui, and K. Qin: Proc. 2015 Int. Conf. Geoinformatics 1. https://doi.org/10.1109/GEOINFORMATICS.2015.7378687
18 R. Garcia, J. P. Castro, E. Verdu, M. J. Verdu, and L. M. Requeras: Cartography – A Tool for Spatial Analysis, C. Bateira, Ed. (Intech, London, 2012) p. 26. https://doi.org/10.5772/46129
19 F. Nah: Behav. Inf. Technol. **23** (2004) 153. https://doi.org/10.1080/01449290410001669914
20 J. T. Sample and E. Loup: Tile-Based Geospatial Information Systems (Springer US, Boston, 2010) p. 17. https://doi.org/10.1007/978-1-4419-7631-4
21 OpenGIS Web Map Tile Service Implementation Standard: https://www.opengeospatial.org/standards/wmts (accessed March 2019).
22 A. Sani and C. Rinner: GEOMATICA **65** (2011) 145. https://doi.org/10.5623/cig2011-023
23 C. Y. Huang and H. Chang: ISPRS Int. J. Geo-Inf. **5** (2016) 136. https://doi.org/10.3390/ijgi5080136
24 C. M. Dalton: Environ. Plann. A: Econ. Space **47** (2015) 1029. https://doi.org/10.1177/0308518X15592302
25 J. Zhang: Ecol. Inf. **8** (2012) 68. https://doi.org/10.1016/j.ecoinf.2012.01.004
26 J. Dean and S. Ghemawat: Commun. ACM **51** (2008) 107. https://doi.org/10.1145/1327452.1327492
27 H. C. Karnatak, R. Shukla, V. K. Sharma, Y. V. S. Murthy, and V. Bhanumurthy: Geocarto Int. **27** (2012) 499. https://doi.org/10.1080/10106049.2011.650651
28 H. Gong, M. Simwanda, and Y. Murayama: ISPRS Geo-Inf. **6** (2017) 257. https://doi.org/10.3390/ijgi6080257