Real-Time Carpooling and Ride-Sharing: Position Paper on Design Concepts, Distribution and Cloud Computing Strategies

Dejan Dimitrijević
Faculty of Technical Sciences
Trg Dositeja Obradovića 6, 21000
Novi Sad, Serbia
Email: dimitrijevic@uns.ac.rs

Nemanja Nedić
Faculty of Technical Sciences
Trg Dositeja Obradovića 6, 21000
Novi Sad, Serbia
Email: nedicn@uns.ac.rs

Vladimir Dimitrieski
Faculty of Technical Sciences
Trg Dositeja Obradovića 6, 21000
Novi Sad, Serbia
Email: dimitrieski@uns.ac.rs

Abstract—Many carpool and ride-sharing solutions have been proposed and even developed in the previous decades, but rarely have they been able to attain a global user base, at least not up until recently. That was mostly because many of them were not initially designed as scalable, leaving their users with a sub-par user experiences as their user base grew, and often their mobile or desktop client reach was not ubiquitous enough, leaving them available only to a small portion of mobile client devices and/or desktop browsers. This paper describes the design concepts, distribution and cloud computing strategies the authors feel any future global carpool and ride-sharing solution could follow, making it very scalable and ubiquitous enough to successfully reach and serve a global user base.

I. INTRODUCTION

THE carpooling, thus also the ride-sharing industry, has only recently started becoming globally interesting. However, carpooling formally appeared in the US in the mid-1970s, after the 1973 oil crisis [1]. At that time the rising costs of using a personal vehicle for transportation of only one passenger made it prudent to drive more than one passenger, usually co-workers commuting daily to and from the same workplace, splitting transportation costs. However, the reduction of oil and gas costs in the 1980s and the breakdown of a typical 9AM to 5PM workday in the 1990s led to a spiral down trend in carpooling popularity. Federal government in the US tried to counter such a trend by giving incentives to carpooling drivers, growing the number of no-toll carpool lanes–the so called, High Occupancy Vehicle (HOV) lanes–across many highways. Those lanes were also allowing for relief from ever growing traffic jams and gridlocks, as the number of vehicles on the roads was ever increasing, which in 2000 exceeded 740 million globally [2] and was projected to be over 2 billion motorized vehicles by 2030 [3]. The sheer number of vehicles alone will create many well-documented problems for urban areas, such as increased traffic, increased pollution, parking congestion, and the need for expensive infrastructure maintenance. To reduce those and also personal transportation costs why not make a global real-time carpool and ride-sharing solution?

A. Problems

As said, the expenses, both environmental and fiscal, of single occupancy vehicles could be reduced by utilizing the empty seats in personal transportation vehicles. Carpooling and ride-sharing target those empty seats: taking additional vehicles off the road reducing traffic and pollution, whilst providing opportunities for social interaction. However, historically carpool scheduling often limited users to consistent schedules and fixed rider groups–carpooling to the same place at the same time with a set person or a group of people. To make that problem worse, the leading problem concerns, given in a 2009 survey about why people don't carpool, were difficulty to organize carpools and inconvenience of organization [4]. We feel both of those can be addressed by employing some novel web technologies and modern day available data stores which hold social and location based individual user's data. Besides having to solve the aforementioned problems for making a carpooling and ride-sharing solution that users will want to use, to make it usable on a global scale the ubiquity problem should also be addressed. By ubiquity we mean the problem of having to make it available across both various mobile and desktop platforms, current and future ones, so our proposed solution also utilizes few other rather novel web technologies.

This paper attempts to propose concepts, distribution and cloud strategies that we feel will bring best value for any future global carpool and ride-sharing solution. The rest of the paper is organized as follows: Section II overviews some related work. Section III gives overall design concepts and our objective. Section IV elaborates on our proof-of-concept prototype system implementation choices, with subsections focusing on several specifics. Section V discusses our future work, plans and intentions and finally Section VI concludes this paper.

II. RELATED WORK

As noted, this section deals with existing carpool related work. Subsection A reviews carpool and ride-sharing related solutions currently available and subsection B surveys some of the literature and papers on the subject.

A. Current carpool and ride-sharing solutions

Entering carpool and ride sharing search terms in some of the largest mobile app store and internet search engines returns a great deal of mobile apps and internet websites offering either classic or dynamic carpooling and ride-sharing. Classic carpool mobile app or website indicates that its users effectively schedule and advertise their plans for a trip well in advance, effectively via a searchable electronic bulletin board, seeking other users travelling in the same direction at the same time either in part or fully. Although some of those apps and websites, such as carpooling.com [6] and its mobile client apps have their uses and large user bases, the static routing problems they help solve makes those uses fairly limited.

The inconvenience of having to search though large carpools or even smaller but fixed choice driver groups, hoping to amongst them find a pre-scheduled and advertised trip adequately consistent with one owns schedule, makes such apps or websites non-practical for relatively short and near-immediate on-the-go carpool and ride sharing trip plans. It is for that reason that even [6] and its large network of European subsidiary websites, added advanced time constrained search features to "find a lift", which to a certain extent alleviate some of the inconveniences for their on-the-go passenger users. However, the added hourly time-constrained advanced searching still inconveniences their vehicle driving users to be mindful of their advertised pre-trip given schedules, even though that may not always be objectively possible, due to unforeseen events such as: road accidents, gridlocks, etc.

Thus, a new form of dynamic carpool and ride-sharing mobile apps and websites is emerging, indicated by their use of real-time passenger requests along with real-time vehicle driving users' location data, foregoing the need for well in advance pre-scheduled and advertised trips. Amongst some of the most known and pioneering mobile apps and websites offering dynamic carpooling and ride-sharing are Lyft [21] and SideCar [22], screenshots of their mobile apps are given in Fig. 1.
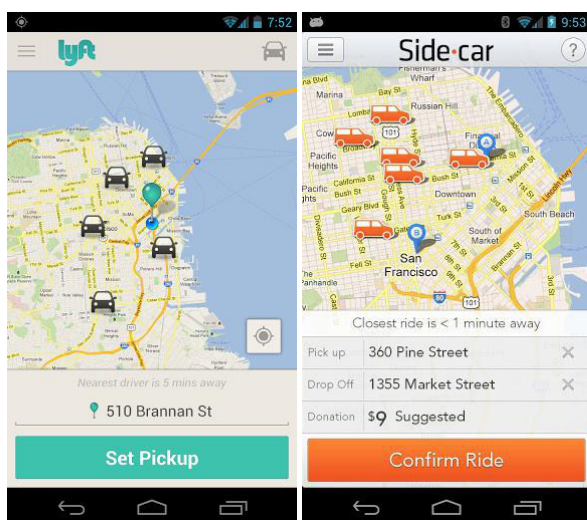
Both of the listed mobile apps are available for iOS as well as for Android mobile platform, from which screenshots are taken, but neither currently present a web browser user interface. This is probably intentional since both mobile apps are natively written, and by our observations both use TCP sockets to communicate with their respective backend services, so they would both need some changes to make them web-friendly. Unfortunately, those code changes could include some rather tedious transformations because native TCP socket traffic has not been well suited for consumption in web browsers relying dominantly on HTTP, until recently. Another popular application and website named Waze [7], which isn't predominantly used for carpool and ride-sharing, but for gridlock traffic reporting and avoidance, seems to have however taken another approach. Although their mobile apps are currently natively written, the website does present a "live map" user interface which maps events reported by their users. Such events include pickup requests and replies of passenger and vehicle driving Waze users who are otherwise linked in a popular social network. Unfortunately, access to those real-time events is currently only provided to its iOS app users and not through backend GeoRSS feed via HTTPS. The original GeoRSS XML format is transformed to JSON for easier web browser JavaScript consumption, and presumably for traffic overhead reduction, but is still limited in update interval time. To our knowledge, there are no other globally popular websites and mobile apps that currently allow for carpool and ride-sharing uses using any other drastically different approaches.

B. Current carpool and ride-sharing papers

Because static variety carpool still represents the majority of existing solutions, almost all of the available papers and literature on carpool and ride-sharing mainly tackle the static ridesharing issues, whereby users must pre-schedule their trips, neglecting the dynamic aspect. Despite much of the progress experimented on dynamic carpooling and ride-sharing concepts thanks to the current solutions, it still remains in the early stages regarding publicly available works and literature that deal with its real-time automation. In order to make up for that shortfall, some of the papers which mention carpooling and ride-sharing, and even some that did considered the dynamic aspect [8], in majority also considered other issues beside the static and dynamic carpooling and ride-sharing problems at the same time. Some papers are especially involved in the concepts of traceability, communication and security services, which their authors feel that none of the current solutions evoked, identifying the security issues as one of the main reasons hindering their success [9]. All cited current papers admittedly still provided us with a lot of beneficial ideas and food for thought transferred onto this paper, its findings and conclusions, and out of that still quite disorganized literature which tackles a lot of issues, we have identified some yet non-tackled, laid out in the following sections. Mainly, we take issue with web browser user interfaces and standardized web technologies which seem to be the unifying way forward, putting ubiquity in the grasp of every hybrid web and mobile application.

Fig. 1 - Screenshots of mobile applications from lyft.me and side.cr

III. Design

In the previous section we cited some of the current solutions, ideas and issues tackled in carpooling and ride-sharing recently. In this section we are building up on those solutions and ideas, proposing some of our own design concepts for a global dynamic real-time carpooling and ride-sharing solution. Subsection A describes some of design concepts we feel are suitable for a real-time dynamic carpooling and ride-sharing solution. Subsection B further extends on A, allowing for the proposed real-time solution to tackle the problem of being able to serve up to a global user base, adding cloud and distribution design concepts. Finally, subsection C tries to deal with the ubiquity problem, considering the client user interface technology we feel will be future-proof and available on almost all new mobile and desktop platforms.

A. Real-time dynamic solution design concepts

As it was noted in section II, real-time dynamic carpooling and ride-sharing solutions are becoming more common amongst the current carpooling and ride-sharing solutions, although it takes more designing effort to achieve real-time dynamic capabilities than for mere static carpooling and ride-sharing. The reason for the recent increase is obviously because real-time dynamic solutions are more convenient, and thus more likely to be used in greater numbers by end users, but also because some technologies previously used for seemingly real-time communication on the web, have only recently matured and have been standardized.

In the begging of the so called Web 2.0, at the time when real-time updating websites were only just starting to appear, most of those websites used Asynchronous JavaScript and XML (AJAX) [10], which is a group of interrelated web development techniques used on the client-side to create asynchronous, seemingly real-time web applications. Most of those techniques relied upon regular HTTP, a simple request-response and stateless protocol. Having to achieve what was usually two-way communication took some effort for websites and web applications, using various workarounds, techniques involving the use of the browser XmlHttpRequest object or some other web browser plugins.

The first workarounds developed into techniques known as: frequent polling, long-polling and the so called forever-frames. Although all of those techniques were, and still are, very much usable for seemingly real-time web page updates without requiring full page refreshes, they had drawbacks. Their primary drawback was, notwithstanding client-side implementation difficulties, the amount of server-side and network resources they consume. The server is either forced to respond to a large number of frequent requests, or it opens up a number of long running responses, which additionally occupy its hardware resources. On the other hand, using workarounds such as various browser plugins, although less network and server-side resource demanding, turned out to be non-practical, because of the lack of plugin support on current mobile devices. For such reasons, new techniques were developed, and recently standardized by the W3C. As part of the HTML5 specification Server-Sent DOM Events (SSE) were standardized in 2011 [11], but have not yet been implemented by all desktop browsers, namely, Internet Explorer. However, Web Sockets API [12], drafted a protocol back in 2009 currently supported by all major web browsers. Web Sockets provide a full-duplex communication channel over a single TCP connection, thus allowing for a lower network latency time due to less traffic overhead compared to HTTP. Compared to SSE and other polling techniques Web Sockets provide the best option for building real-time communication on the web, and that is why such a protocol is part of our proposed design concept.

B. Distribution and cloud design concepts

Having chosen Web Sockets (WS) as a preferred means of communication, although helping solve latency issues which can lead to a great number of performance problems in building real-time solutions, left another issue unsolved. WS based communication, as all others, still has a limit on the maximum number of simultaneous clients connected to a single server node. Even though that number may be greater when using WS, it still depends on available server hardware resources. Since vertical scaling of server hardware resources can be expensive and still limiting, the solution to the problem is horizontal distribution, across multiple server nodes. Ideally, any global real-time solution would be best served in one's own server farm, but given hardware and its maintenance costs, renting cloud resources works as well. However, for horizontal scaling, one needs to be able to scale data also. Since traditional, i.e. relational data scaling is much harder [13], we have turned to non-relational data (NoSQL). NoSQL databases besides easier scaling, offer better performance in data writes, as well as a possibility of scaling reads onto multiple database nodes, combining sharding and some parallelism approaches. Utilizing the two in a document oriented NoSQL data store that supports geospatial data indexing would make it a perfect fit for our proposed solution and storing our users' location based data. Also, a key-value memory caching NoSQL data store could be used as a messaging backplane for communication between our individual server nodes, but that use is trivial.

C. User interface architecture design concepts

To make a website or mobile app truly ubiquitous, one needs to support as many different desktop and mobile platforms as possible, ideally all of them. Although client applications can be natively written for each platform, there are some unifying user interface technologies for almost all current desktop and smartphone mobile platforms today.

So, to achieve ubiquity, we propose the use of combined HTML5/CSS3 for user interface (UI) rendering. Building the UI around streamed real-time data flows of state changes created by passenger and vehicle driving user events (users requesting rides, user driving busy, user driving free, etc.) is why the paradigm of reactive programming seems to be the perfect choice. Reactive programming is not to be confused with responsive web design, which is also utilized, for same UI reuse across various device screen resolution sizes.

Any changes in state registered by user client applications are asynchronously transferred via WS to distributed load balanced cloud server nodes which are displayed in Fig. 2.
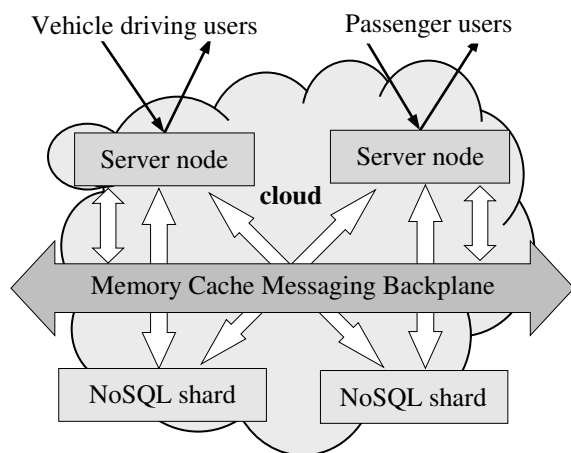

Fig. 2 - Basic design diagram

IV. PROTOTYPE IMPLEMENTATION

This section describes in more detail some of the implementation choices we used to build the prototype of our distributable cloud-based dynamic real-time carpooling and ride-sharing solution. Subsection A describes our prototype's real-time communication transport library choice. Subsection B deals with our use of NoSQL data stores for geospatial indexed data and fast memory caching messaging backplane implementation, along with our choices of NoSQL technology and products used. Subsection C goes into some of our UI implementation details.

A. Real-time communication

Having helped develop the first online taxi dispatching solution in Serbia, realized in .NET and being cloud-hosted on Microsoft Azure [14], influenced a lot of our primary technology choices for the prototype of a real-time dynamic carpooling and ride-sharing solution to be described here.

As noted in the design section, the need to have a real-time dynamic carpooling and ride-sharing solution is imperative, since those solutions are what most users currently wish to use. To make our prototype solution real-time capable, the choice to implement it using a library capable of WS protocol communication in .NET came down to a library named SignalR [15]. SignalR is an open-source library for ASP.NET to add real-time web functionality to .NET applications, adding the ability for server-side code to push content to the connected clients as it happens, in real-time. SignalR server is capable of supporting clients written in .NET, JavaScript and some other programming languages. The server-side code can push content to those connected clients via a number of transport techniques, most suitable being bi-directional WS, if available. For a server-side the WS transport requirements are either a self-hosted ASP.NET

4.0 application or one hosted within Internet Information Services (IIS) 8. Server-side hosted on earlier versions of IIS fallbacks to other means of message transports. For clients, WS requirement issue is a bit lengthier to describe, so it is listed in better detail in the client UI subsection.

B. NoSQL implementation

Since IIS 8 was our prototype's hosting platform of choice, it should be noted that the server node could then only have been hosted within the Windows 8 / Server 2012 OS platforms. Fortunately, Windows Server 2012 was made available to end-users on the Windows Azure cloud platform as a virtual machine operating system choice since late 2012, and it is deployable onto an Extra Small machine instance. Extra Small Windows Azure instance, which entails a shared core processor with only 768MB RAM, may not be the first choice from a performance standpoint. However, it is sufficient for proof-of-concept deployments and for limiting cloud-hosting costs.

Deploying SignalR server-side code alongside a NoSQL key-value memory cache data store named Redis [16], with the minimum amount of RAM allocated to Redis, produced a fully functioning server node capable of serving a test number of simultaneous users. Since a new server node can be cloned, and any cloned node's Redis instance can then be easily subscribed to a Redis instance of an existing node, we can easily increase the number of new server nodes to meet all of our scaling needs. Scale out is so easily achieved in part due to SignalR's in-built scaling mechanisms, which uses Redis pub/sub features for a messaging backplane. Each SignalR server node could then, through its Redis instance, be notified of any new WS communication channel needed in real-time. The load balancer of connected computing cloud instances, which is built into Windows Azure, takes care of diverting traffic to a SignalR server node best able (least busy) to process any incoming new or reoccurring real-time request. But since each node has then been notified, by its Redis instance, that such communication has taken place each channel should be reusable by any other SignalR node, so each node is capable of replying to any real-time request.

Beside Redis NoSQL, our prototype incorporated another NoSQL document-oriented geospatial indexing data store for ease of scaling, named MongoDB [17]. MongoDB allows for rapid storing of user current locations by extensions to the stored location data, incorporating another key value used for sharding of that data across multiple MongoDB store instances. Data sharding approach allows for quicker reads since our shard key represents a geographical area within which our users are seeking or soliciting ride request during their relatively short trips. Thus, sharding increases read performance by reducing the amount of indexed data being queried in each MongoDB instance, because that data gets spread out across multiple data store instances. By adding MongoDB's built-in MapReduce (MR) to our queries the tasks of searching through sharded data execute in parallel.

Also, MongoDB by itself incorporates some replication set mechanisms, giving it a highly-available aspect as well, which bodes well with the SignalR's ability to distribute via Redis as a messaging backplane, reducing single points of failure.

C. User interface

Finally, for ubiquity reasons, our choice of prototype client UI rendering technology incorporated HTML5/CSS3. Having previously built a fully functional HTML5/CSS3 client for a commercial online taxi dispatcher, which was wrapped as an app using a mobile development platform named PhoneGap [18] runnable across various mobile platforms, we felt confident that HTML5/CSS3 was also a right choice. Both desktop and mobile web clients shared the same JavaScript logic codebase which offered unified access to geolocation [19] features of the devices they all ran on, a must for a dynamic carpooling and ride-sharing applications. The look and feel across smaller resolutions changes accordingly, but not drastically, by utilizing responsive CSS3 design incorporated in jQuery mobile [20] as of version 1.3.

All the clients also use a reactive programming paradigm, connecting to the backend via code using the Reactive extensions for JavaScript (RxJS) library [21]. This means the user interface responds asynchronously to user actions and events which they result as, either events which are streamed from the server-side generated by other users and fed via WS to all supporting clients, or user's own events. If, per chance, the mobile device's web browser does not support WS transport, SignalR client in JavaScript will gracefully fall back to other means of seemingly real-time transports, which RxJS will still continue to process as asynchronous events.

Support for WS as a mean of communication transport, depends primarily on a platform web browser's capabilities which is for current desktop and mobile web browsers given in Table I.

TABLE I.
WEB BROWSER SUPPORT FOR WEBSOCKETS

| Web browser	Supported since version	Supported
Internet Explorer	10.0 (fully)	Yes
Firefox	4.0 (partially) 6.0 (fully)	Yes
Chrome	4.0 (partially) 14.0 (fully)	Yes
Safari	5.0 (partially) 6.0 (fully)	Yes
Opera	11.0 (partially) 12.1 (fully)	Yes
iOS Safari	11.0 (partially) 12.1 (fully)	Yes
Opera Mini	-	No
Android Browser	-	No
BlackBerry Browser	7.0 (fully)	Yes
Opera Mobile	11.0 (partially) 12.1 (fully)	Yes
Crome for Android	25.0 (fully)	Yes
Firefox for Android	19.0 (fully)	Yes
Firefox OS Boot2Gecko	1.0.0-prerelease (fully)	Yes
Tizen OS	2.0.0a-emulator (fully)	Yes
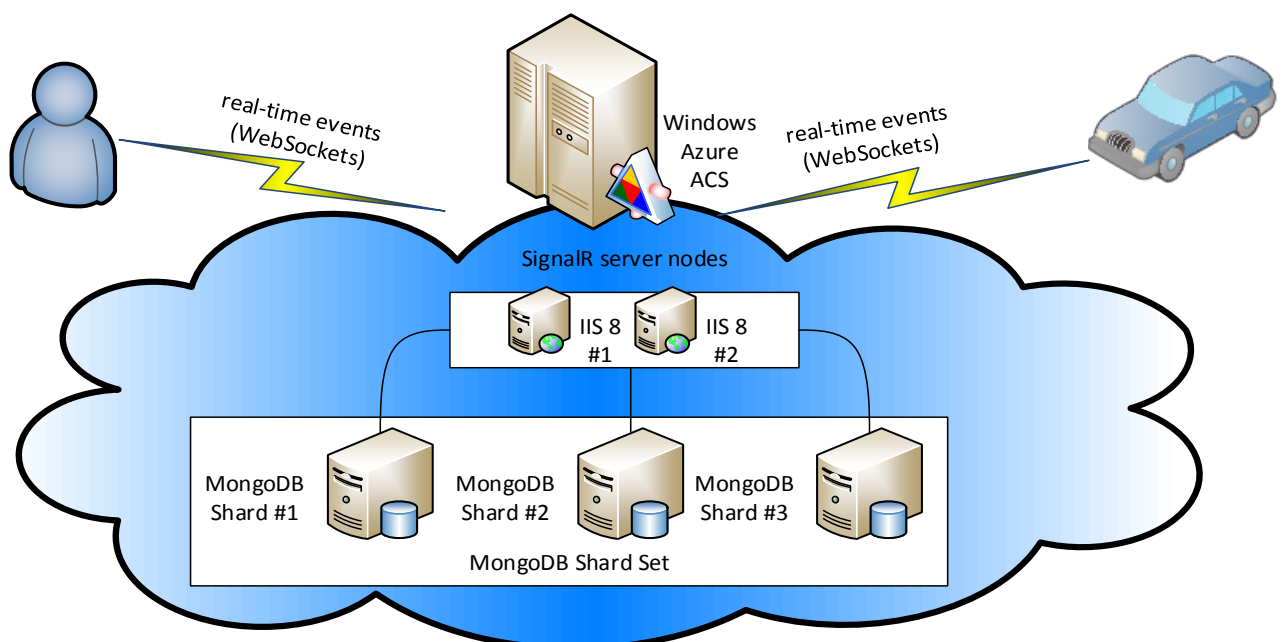
Figure 3 – Windows Azure hosted prototype design diagram displaying some implementation details

V. Future Work

Having described some our prototype's implementation details (Fig. 3) our future work and plans envision for it to be deployed and tested in the real-world. Since the prototype clients were based on previous work done for a commercial online taxi dispatcher, it will be initially tested and deployed as part of that solution in limited numbers. Early adopters of the online taxi dispatching service will get the benefit of being able to track a few assigned taxis in real-time. Drivers of those taxis will be either issued mobile devices with pre-installed HTML5/CSS3 web clients and/or those client apps will be installed on their own devices. Such real-world tests will hopefully lead to identifying problems not yet foreseen. Once a stable solution is reached the prototype could and will become a standalone service, open for public use and not just for taxi dispatching and the cost of its operational maintenance could then also be better estimated. If deemed low enough to be offset by ad support according to [20], its use could be completely free for end users unlike [21, 22].

To reach that point however, some other issues, such as security and privacy, will also need to be tackled. In [5] the solution for the security and privacy issue was implied by use of a 3rd party location based service (LBS), which used OAuth protocol to authenticate and subsequently authorize which exact set of users would be allowed access to the authorizing user's location. Unfortunately, come February 2013 the 3rd party LBS was shut down, and an alternative solution should either be found or developed prior to prototype's launch as a standalone service. Trying to avoid the repeat of having to find alternatives to a 3rd party components not being operational any more, the focus could be on building up own LBS features respective of privacy, but relying on information which can be provided from popular social networks. To aid us in that endeavor, instrumental part of the puzzle could be Windows Azure built-in Access Control Service (ACS), allowing for users to single sign-on to the proposed carpool and ride-sharing service just as if they were signing into the aforementioned social networks. If those users comply, their location data could then only be made accessible to a subset of their social network friends, a widely acceptable solution from a current privacy standpoint.

VI. Conclusion

This paper tried to underscore the need for developing dynamic real-time carpool and ride-sharing solutions, instead of already outdated static ones, by employing some novel web technologies and approaches. Since a prototype has been successfully developed following the outlined design concepts, distribution and cloud strategies, it is obviously possible to build other such solutions using the same approaches. Especially interesting is the possibility to develop a web platform application that runs across multiple devices and their web browsers, be they mobile or desktop.

Using an open-source jQuery mobile library and Apache Cordova [23] mobile developer platform, which was derived from PhoneGap, is what interests us the most and we feel could be the unifying tools for any future service supposedly usable across multiple operating systems, current and future. Combining those with some other frameworks which use the HTML5 UI elements such as the canvas tag thus adding the ability to render graphical data such as street level maps for carpool should, by our position, be the leading way forward.[i]

VII. References

[1] Ozanne, L., & Mollenkopf, D. (1999). "Understanding consumer intentions to carpool: a test of alternative models." In Proceedings of the 1999 annual meeting of the Australian & New Zealand Marketing Academy. smib.vuw.ac.nz (Vol. 8081).

[2] Fraichard, T. (2005). "Cybercar: l'alternative à la voiture particulière." Navigation (Paris), 53(1), 53-74.

[3] Dargay, J., & Hanly, M. (2007). "Volatility of car ownership, commuting mode and time in the UK." Transportation Research Part A: Policy and Practice, 41(10), 934-948.

[4] Massaro, Dominic W., et al. (2009) "CARPOOLNOW: Just-in-time carpooling without elaborate preplanning." the 5th International Conference on Web Information Systems and Technologies. Lisbon, Portugal. 2009.

[5] Dimitrijević, D., & Luković, I., & Dimitrieski, V., & Vasiljević, I. (2013) "Orchestrating Yahoo! FireEagle location based service for carpooling" 3rd International Conference on Information Society Technology and Management, Kopaonik, Serbia, 2013.

[6] The largest car sharing network for cheap, green travel in Europe. Web - carpooling.com

[7] Outsmarting traffic, together. Web - waze.com

[8] Sghaier, M., Zgaya, H., Hammadi, S., & Tahon, C. (2011). A Distributed Optimized Approach based on the Multi Agent Concept for the Implementation of a Real Time Carpooling Service with an Optimization Aspect on Siblings. International Journal of Engineering (IJE), 5(2), 217.

[9] Sghaier, M., Zgaya, H., Hammadi, S., & Tahon, C. (2010, September). A distributed dijkstra's algorithm for the implementation of a Real Time Carpooling Service with an optimized aspect on siblings. In Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on (pp. 795-800). IEEE.

[10] Garrett, J. J. (2005). Ajax: A new approach to web applications.

[11] Hickson, I. Server-Sent Events, W3C Working Draft 20 October 2011.

[12] Hickson, I. (2010). The Web Sockets API, W3C Working Draft 29 October 2009.

[13] Cattell, R. (2011). Scalable SQL and NoSQL data stores. ACM SIGMOD Record, 39(4), 12-27.

[14] Najbrži put do slobodnog vozila. Web - taxiproxy.com

[15] ASP.NET SignalR : Incredibly simple real-time web for .NET. Web – signalr.net

[16] Redis. Web – redis.io

[17] MongoDB. Web – mongodb.org

[18] PhoneGap. Web – phonegap.com

[19] Popescu, A. (2010). Geolocation api specification. World Wide Web Consortium, Candidate Recommendation CR-geolocation-API-20100907.

[20] Goldstein, D. G., McAfee, R. P., & Suri, S. (2013, May). The cost of annoying ads. In Proceedings of the 22nd international conference on World Wide Web (pp. 459-470). International World Wide Web Conferences Steering Committee.

[21] Lyft. Web – lyft.me

[22] SideCar. Web – side.cr

[23] Apache Codrova. Web – cordova.apache.org

---

[i] This is to acknowledge all of the not so small help of our PhD advisors.