2011 Fourth IEEE International Conference on Utility and Cloud Computing

Participatory Cyber Physical System in Public Transport Application

John Kah-Soon Lau, Chen-Khong Tham, Tie Luo
Department of Electrical and Computer Engineering
National University of Singapore
{elelksj, eletck, eleluo}@nus.edu.sg

***Abstract*—Population in the urban areas around the world is increasing yet energy and natural resources which are non-renewable, have continued to deplete. This scenario leads us to the many challenges faced by public transportation since it is directly linked to urbanization. From a civilian point of view, proper travel planning and knowledge of others' travel experiences are useful in easing public transportation woes. We apply Cyber Physical System (CPS) within cyber-physical-socio space to make those capabilities possible. In this approach, we regard data as a service, data are contributed and queried by the public masses. Location and time represent the physical domain and data with information derived from them represent the digital domain in our CPS design. We designed and implemented a prototypical system named ContriSenseCloud that works based on client-server model. The system has two main distinctive platforms, namely Service Exchange Platform (SEP) and Application-specific Exchange Platform (AEP). The SEP is designed to be as generalizable as possible whereas AEP is where our current public transportation applications reside. Such architecture enables extensibility of ContriSenseCloud to other application domains. Other contributions include RESTful Application Programming Interface (API) in CPS, near real-time sensing, and mapping of GPS readings to correct sequence of bus stops.**

***Index Terms*—Cyber physical system, cyber physical society, participatory sensing, data as a service, public transportation**

I. Introduction

To encourage public transport ridership remains a main agenda in counteracting irreversible human impact behind climate change [1]. However, the population density increase at large cities has made the effort more difficult. Peak hour during which high level of passenger congestion and traffic congestion often occur, is usually predefined though it may actually vary across different locations. Being able to observe such variation in real-time will add value to commuters' travel experience so that they can plan their journey accordingly. However, providing such a capability is faced with the challenge that either transit agencies do not deploy real-time tracking [2] or they are reluctant to release the data which they deem proprietary.

Hence we present ContriSenseCloud, an extensible cyber physical system that manages grassroots participatory sensing in public transportation. The first application ContriSenseCloud hosts is ContriSense:Bus, which helps commuters plan their bus journey based on information distilled from user-contributed data and allows them to contribute data via smartphones. The system also allows third party developers to access anonymized user-contributed spatial-temporal data as a web service through RESTful [3] API.

Participatory sensing makes use of mobile devices to create interactive, participatory sensor networks that allow both public and professional users to collect, analyze, and share local knowledge [4]. A cyber physical system is an integration of physical processes and digital computation that exposes its computing capability to the outside by networking. Without such exposure, it is merely an embedded system [5]. The system can be extended into cyber-physical-human space or cyber-physical-socio space. Cyber-physical-socio space or cyber physical society implies merging of social interaction between humans with physical processes and digital computation whereas cyber-physical-human space lacks the interaction [6]. Our system design shows that participatory sensing concept is closely related to cyber-physical-human space or cyber-physical-socio space.

Previous implemented participatory sensing systems via smartphones focused on various classifications and method to track underground vehicle [2] as well as on personal pattern discovery [7]. However our focus is to deploy participatory sensing application in CPS that delivers data as a service. We apply GPS trajectory database and MATE algorithm [8] in developing ContriSense:Bus application. The TPF system was evaluated with simulation based on real-time actual traffic data [8] while ContriSenseCloud and ContriSense:Bus were productized for public usage. In context of [6], E. Lu et al implemented system is within cyber-physical space whereas our implemented system is within cyber-physical-socio space. We regard a node in GPS trajectory database as a bus stop instead of road intersection and GPS trajectory to navigation path mapping is direction dependent instead of directionless. Another past related work is the development of CPS as an application framework. The framework is accessible by Java library classes, in other words code as API [9]. In our case, ContriSenseCloud delivers data as API. CPS based on cyber-physical-socio space named SenseWorld was implemented and demonstrated [10]. SenseWorld did not use smartphone as sensor though the paper stated that it would be incorporated as part of future work.

In the next section, the design and implementation of ContriSenseCloud is described. Section III mentions how RESTful API works in the system. Section IV contains ContriSense:Bus algorithms, and Section V, preliminary system evaluation.

978-0-7695-4592-9/11 $26.00 © 2011 IEEE
DOI 10.1109/UCC.2011.59

Section VI concludes the paper.

II. System Architecture

In our system, data and information are different concepts in that the latter are derived from processing and analyzing the former. For example in ContriSense:Bus, data are user-contributed GPS with timestamp traces and information are segments of travel time between 2 different places. Bus commuters represent the public users and third party developers represent the professional users with regard to participatory sensing definition. Furthermore, public users are either contributors who are compelled by an incentive scheme to contribute data regardless of travel experience quality or inquirers who benefit from contributed data in order to avoid undesired journey at certain point in time.

The system design is based on client-server model in which the client-side consists of smartphones and static sensor systems (e.g. temperature, motion, sound) whereas the server-side runs on top of JBoss Application Server (AS) middleware that connects to MySQL database. In Figure 1, there are two major platforms:-

- Service Exchange Platform (SEP) - General platform that deals with data and information exchange.
  - ◇ Service Discovery - Enumerates all services hosted.
  - ◇ Service Consumer - Serves data as an API.
  - ◇ Service Contributor - Allow developers to stream or store their data.
  - ◇ Event Processing - Process incoming data e.g. averaging, summing, counting, etc.
- Application-specific Exchange Platform (AEP) - Specialized platform that serves various applications e.g. ContriSense:Bus and ContriSense:Car.

SEP and AEP are related to each other in the sense that applications that are developed using data as API from SEP can later be hosted on AEP. Developers have a choice of hosting applications themselves or on AEP. SEP is designed to be as generic as possible so it can scale up as more modalities are added in. It is generalizable because all incoming and outgoing data are declared as string data type. Since there is only one data type, new modalities code can be auto-generated before recompilation.

In Figure 2, spatial-temporal sensing is periodic event and query is aperiodic event [11] between Android client and the server. Spatial-temporal sensing starts from client-side in which GPS with timestamps are recorded every 15 seconds. Each record is stored in memory and sent to server after 3 minutes. Therefore there are 12 records gathered in each 3 minutes period. Query on the other hand can be executed at any point in time even while sensing is running. The C++ daemon performs more intensive computations on behalf of stateless PHP programs. Apache Thrift [12] is used for seamless binary communication between PHP and C++ programs.

Illustrated in Figure 3, supply and demand between users and developers form social interaction in which the system acts as intermediary. Supply is user-contributed data and demand is either information queried by users or data subscribed

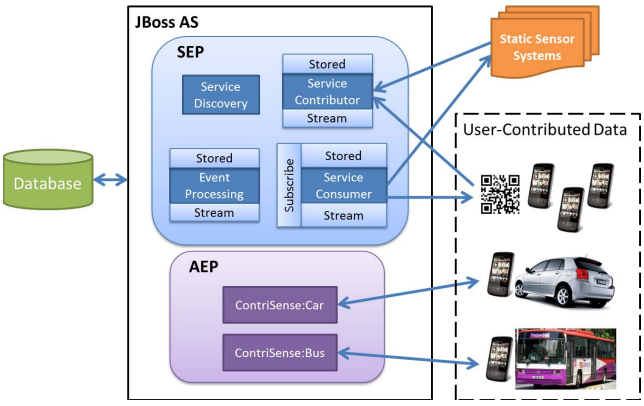

Fig. 1. ContriSenseCloud architecture (Arrow indicates data and information flow)

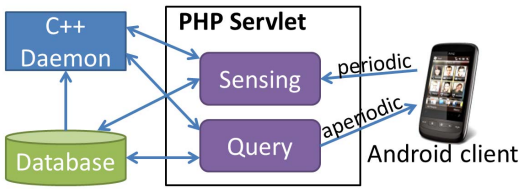

Fig. 2. ContriSense:Bus architecture

by developers. By removing supply-demand and subscription (service consumer), the system loop becomes opened and it is now within cyber-physical-human space instead of cyber-physical-socio space.

III. RESTful API

The SEP is a Java servlet developed using RESTEasy framework that is in full compliant with JAX-RS (Java API for RESTful Web Services) specification. We use ContriSense:Bus as an example to briefly describe how the API works for developers. Based on Figure 3, assume the following:-

- Commuter C and commuter D contribute spatial-temporal data to ContriSense:Bus with their smartphones while traveling on buses.
- In SEP, spatial-temporal data from buses are registered with service ID number 1.

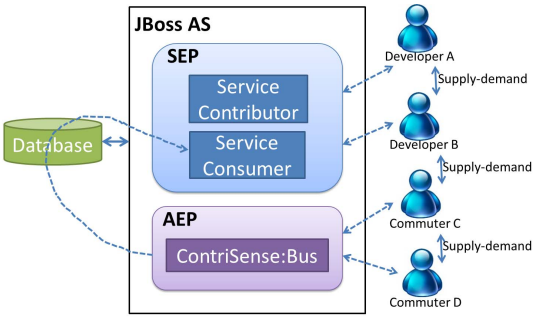

Fig. 3. Cyber physical society

- Trace ID is 1 for commuter C and 2 for commuter D.
- Trace ID number 1 has 20 spatial-temporal data i.e. GPS points with timestamps and trace ID number 2 has 30 spatial-temporal data.
- The base URI for the API as a web service is `http://example.com/xchange/api/data/view`.

The data or resources are represented in XML and JSON format of which developers have a choice of selecting either one. Developers can write applications to send requests to SEP via HTTP in order to obtain historical (stored mode) or latest (stored or steam mode) data. Currently spatial-temporal data are stored in MySQL database and to access them, one can use HTTP GET method such that `http://example.com/xchange/api/data/view/[service ID]/[trace ID]/[trace index]` hence:-

- To access the whole current spatial-temporal data
  - ◇ XML:
    `http://example.com/xchange/api/data/view/1`
  - ◇ JSON:
    `http://example.com/xchange/api/data/json/view/1`
- To access trace ID number 2
  - ◇ XML:
    `http://example.com/xchange/api/data/view/1/2`
  - ◇ JSON:
    `http://example.com/xchange/api/data/json/view/1/2`
- To access 18^{th} datum (a point) of trace ID number 1
  - ◇ XML:
    `http://example.com/xchange/api/data/view/1/1/18`
  - ◇ JSON:
    `http://example.com/xchange/api/data/json/view/1/1/18`

We also provide RESTful features for users to stream, store and subscribe other form of data such as temperature, sound, et cetera. We do not elaborate on this since it is beyond the scope of this paper.

IV. ContriSense:Bus Algorithms

We developed near real-time sensing because transmitting each spatio-temporal datum in real-time to server consumes a lot of client-side devices battery power. Therefore the datum is collected every 15 seconds instead of 1 or 2 seconds to reduce client power consumption and server load. Table I outlines the aforementioned algorithm. As for querying in ContriSense:Bus, the information generated from user-contributed data are considered real-time only if there are at least a few users contributing data at the queried bus service, its direction, and time. NRTS algorithm is implemented in PHP and S2B algorithm is implemented in C++.

As depicted in Figure 4, the red line along the road is the actual bus trajectory on which a user is contributing while commuting whereas the green straight line connects at least two successive bus stops mapped from the actual trajectory. The green line consists of segment(s) with each segment defined as a line that connects 2 nodes or bus stops. The objective is to prevent the black line from been plotted as a segment since it violates the predefined bus stop sequence

TABLE I

NRTS: Near Real-time Sensing Algorithm
(Client-side) 1. $T = N\tau$ where $T = 180s$ and $\tau = 15s$ hence $N = 12$. 2. Record $d = [i, j, k, l, m, n, o, p]$ at each τ. 3. Every $t = t + T$, send $S = \{d_1, d_2, ..., d_N\}$ to server via HTTP. (Server-side) 1. Accept d from client. 2. Check if the trip is registered in database by i. 3. IF i does not exist:- a. Insert i, j, k, l, m, n, o, p into database. b. $node_{ID}$ = S2B(j, k, o, p, 0). Refer to Table II. c. Store $node_{ID}$ in database. 4. ELSE:- a. $node_{ID}^{previous}$ = $node_{ID}$ from 3.c or 4.d.vii. b. Insert i, j, k, l, m, n, o, p into database. c. $node_{ID}$ = S2B(j, k, o, p, $node_{ID}^{previous}$). d. IF $node_{ID}$!= $node_{ID}^{previous}$ (means a segment is detected):- i. Check whether the segment exists in database by $node_{ID}$ and $node_{ID}^{previous}$. ii. IF segment does not exist:- 1. Assign unique number to $segment_{ID}$. 2. Insert $node_{ID}$ and $node_{ID}^{previous}$ tagged to $segment_{ID}$ in database. iii. ELSE:- 1. Retrieve $segment_{ID}$ from database. iv. $\Delta t_{segment_{ID}} = m_{node_{ID}} - m_{node_{ID}^{previous}}$ v. $t_c = \frac{m_{node_{ID}}(HH \times 3600 + MM \times 60 + SS)}{900s}$ vi. Store $\Delta t_{segment_{ID}}$ and t_c tagged to $segment_{ID}$ for MATE algorithm. vii. Update $node_{ID}$ and x tagged to i in database. e. ELSE:- i. Update x tagged to i in database.
t = recording time. i = unique trip ID. j = latitude. k = longitude. l = GPS point radius (an accuracy indicator). m = timestamp (YYYY-MM-DD HH:MM:SS). n = username. Not accessible via SEP. o = bus service number. p = bus service direction. T = period of uploading S to server. τ = period of getting d. d = a spatial-temporal datum. S = set of d per period of T. Segment = a pair of nodes with different ID string. $\Delta t_{segment_{ID}}$ = travel time of a segment. t_c = time code, representation of time in 900s (15 minutes). Trace = the whole spatial-temporal data in 1 journey. = $\{S_T, S_{2T}, S_{3T}, ..., S_{total\ travel\ time}\}$. = $\{d_x\}$ where x = trace index number $= 0, 1, 2, ..., \lvert trace \rvert - 1$

Fig. 4. Application of S2B algorithm

TABLE II

S2B: Spatial Data to Bus Stops Mapping Algorithm
1. Get B_S from memory with S as the key. 2. Initialize d_{min} to a large number. 3. IF j and k are the first coordinate:- a. FOREACH ix in B_S:- i. $d_{gps} = haversine((j,k),(j(B_S[ix]),k(B_S[ix])))$ ii. IF $d_{gps} < d_{min}$:- 1. $d_{min} = d_{gps}$ 2. IF $d_{min} <= K_{max}$ THEN $node_{ID}^{min} = B_S[ix]$ 4. ELSE:- a. FOREACH ix in B_S:- i. IF continue_flag == TRUE:- 1. $node_{ID}^{next} = B_S[ix]$ 2. $d_{gps} = haversine((j,k),(j(B_S[ix]),k(B_S[ix])))$ 3. $d_r = d_{gps} \exp(ix - ix_{current})$ 4. IF $d_r < d_{min}$:- a. $d_{min} = d_r$ b. IF $d_{min} <= K_{max}$:- i. $node_{ID}^{min} = node_{ID}^{next}$ ii. IF $d_{min} <= K_{min}$ THEN BREAK ii. IF $B_S[ix]$ == $node_{ID}^{previous}$ AND ix != $\|B_S\| - 1$:- 1. $ix_{current} = ix$ 2. $node_{ID}^{min} = node_{ID}^{previous}$ 3. Set continue_flag = TRUE 5. Return $node_{ID}^{min}$
j = latitude. k = longitude. o = bus service number. p = bus service direction. d_r = relative distance. d_{gps} = haversine distance. d_{min} = minimum distance. K_{max} = maximum scaling constant = 0.5 K_{min} = minimum scaling constant = 0.15 $node_{ID}^{min}$ = closest correct node ID. $node_{ID}^{previous}$ = previous node ID. $node_{ID}^{next}$ = next node ID. S = o concatenated to 'd' and p. e.g. 151d0 or 151d1, to simplify data structure. ix = bus stop index number. B_S = bus stop sequence of a particular bus service and direction. =$\{node_{ID,ix}\}$ where $ix = 0, 1, 2, ..., \|B_S\| - 1$

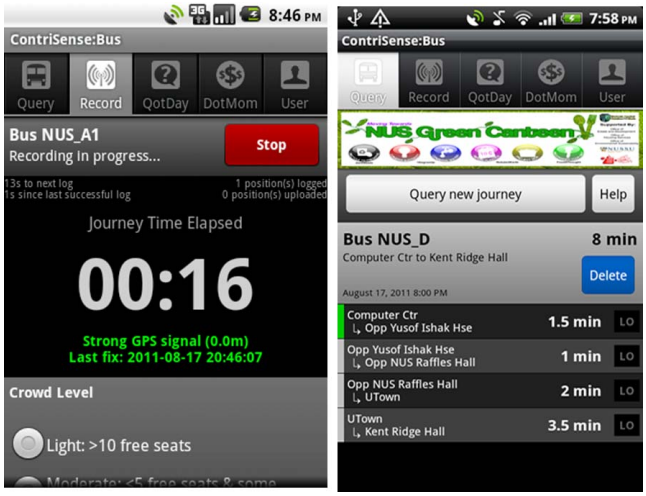

Fig. 5. Android app screenshots

even though the 2 bus stops are nearest to each other. To ensure the correct green line is indexed as a segment, the S2B algorithm (Table II) does not only compute haversine distance between 2 coordinates to determine the next bus stop but also the exponential distance of the next bus stop index number to current bus stop index number in an ordered bus stop sequence list. In other words, the further away a bus stop is to the current bus stop in a predefined sequence regardless of physical distance, the higher weight assigned. Our current system supports all SBS and SMRT public bus services as well as NUS shuttle bus services in Singapore in which their predefined bus stop sequences were gathered from [13], [14].

V. Preliminary System Evaluation

We have released ContriSense:Bus Android application to Android Market and as of September 2011, there are 246 installs [15] and 148 user sign-ups. Figure 5 is the captured screenshots of sensing and querying on Android smartphones, where there are 4 segments listed in the query result.

Shown in Figure 6 are 4 traces contributed by users on different bus services around Singapore with bus stops correctly mapped from spatial trajectories. The traces are visualized by using Google Maps Javascript API. Nevertheless there are conditional cases in which an additional bus stop was erroneously mapped because it is very near to the destination bus stop at which users alighted. Also, the additional bus stop is the next predefined stop directly after the destination bus stop. Based on traces received so far, such errors are uncommon as most bus stops are situated much further from each other. Identified root cause is that K_{max} and K_{min} are not dynamic.

The system server-side has been deployed and evaluated on Dell PowerEdge T410 with CPU (2.4GHz, 12MB cache), 12GB RAM, and 1.0Gbps network speed. The evaluation is divided into 2 parts which are performance measurements between Android clients and server as well as between developers' API call and server. The measurements were done through siege [16], an open source HTTP/HTTPS stress tester.

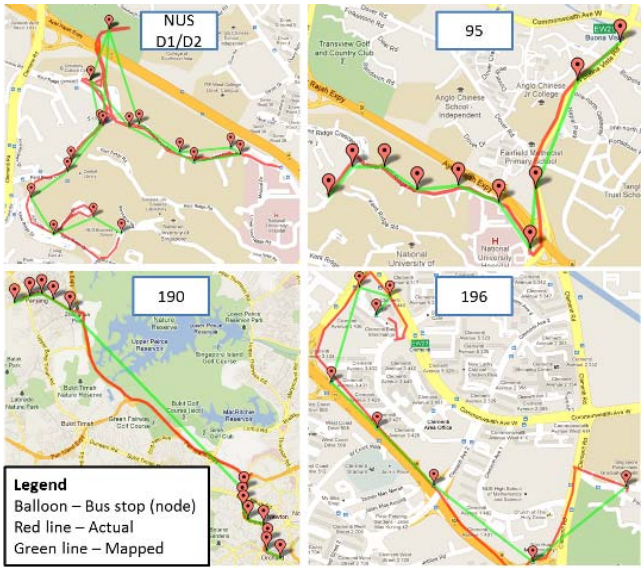

Fig. 6. User-contributed traces

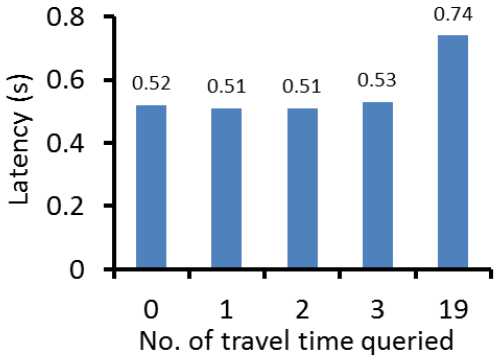

Fig. 7. Test case 1 result

Latency or response time from server was measured in 4 test cases. All test cases except test case 3 were based on only 1 actual user-contributed trace (ID: 29, Bus: 151, Direction: 0, From: Bef Crown Ctr, To: Opp Blk 352, Start time: 2011-08-29 17:28:48, End time: 2011-08-29 17:46:07, Content: 57 recorded spatial-temporal data, Number of segments: 19). Using siege tool, each test case was repeated for 30 seconds to calculate the average response time.

A. Android Clients and Server

Test case 1 assumed 1 user querying the server for travel time between 2 bus stops. The user submitted 5 queries in which each corresponding result gave 0 (for 1 segment with travel time = 0 that means no data contributed yet), 1 (for 1 segment), 2 (for 2 segments), 3 (for 3 segments), and 19 (for 19 segments) travel durations. The test case result on Figure 7 shows that latency is directly proportional to number of segments in a query regardless of travel time value since

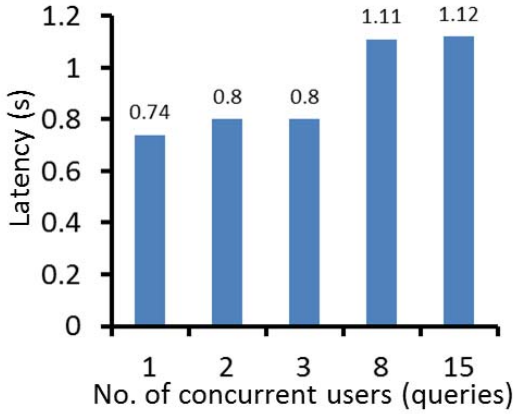

Fig. 8. Test case 2 result

```
<spatialtemporal>
  <crowd>2</crowd>
  <elev>35</elev>
  <lat>1.324217319488487</lat>
  <lng>103.81007194518742</lng>
  <radius>9.487171</radius>
  <traceId>29</traceId>
  <traceIndex>2</traceIndex>
  <traceTime>2011-08-29 17:29:19.0</traceTime>
</spatialtemporal>
```

XML

```
{"spatialtemporal":
{"crowd":2,"elev":35,"lat":1.324217
319488487,"lng":103.81007194518742,
"radius":9.487171,"traceId":29,"tra
ceIndex":2,"traceTime":"2011-08-29
17:29:19.0"}}
```

JSON

Fig. 9. Anonymized spatial-temporal datum

the latency for 1 queried segment with 0 travel time is near to 1 queried segment with 1 travel time.

In Figure 8, test case 2 validated that query latencies with regard to different number of concurrent users do not differ much because of the non-blocking server feature applied through Apache Thrift libraries. Nevertheless, response time for each query increases with number of concurrent users. Between 3 and 8 concurrent users, the latency difference was larger due to resizing of thread pool whereas between 8 and 15 (or 1 and 3) concurrent users, the latency difference was smaller since the handling of queries was still within thread pool capacity.

B. Developers and Server

Figure 9 highlights the difference between XML and JSON response at API call. Test case 3 proved that response time for JSON is faster than XML due to it being smaller in size. The benefit becomes apparent when the data accessed become larger as observed on 84 traces bar at Figure 10. In test case 3, 1 datum and 1 trace were from aforementioned trace number 29 whereas 84 traces consisted of actual traces contributed by

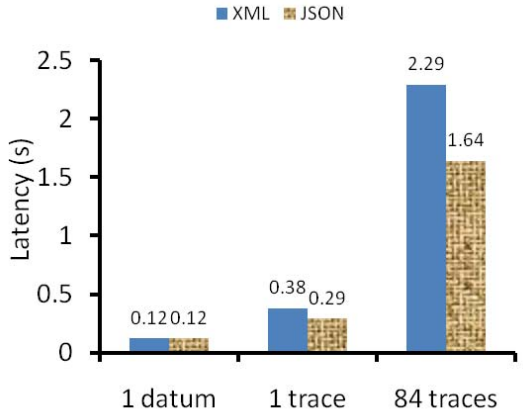

Fig. 10. Test case 3 result

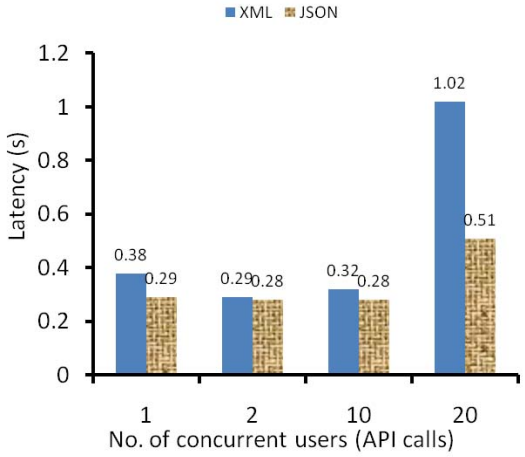

Fig. 11. Test case 4 result

users on various bus services across Singapore.

Test case 4 further affirmed the difference between JSON and XML in terms of performance when the number of concurrent API calls for 1 trace were increased. As seen on Figure 11, latency for JSON remains lower than XML when more API calls were executed concurrently.

VI. Conclusion

We designed and developed a participatory cyber physical system in public transportation that firstly targets bus commuters. The system consists of two main platforms namely SEP and AEP in which SEP serves data as an API or a service to developers whereas AEP serves applications such as ContriSense:Bus to public users. We also proposed NRTS and S2B algorithms which complement GPS trajectory database and MATE algorithm in actual public usage. S2B algorithm has proved useful in translating bus spatial trajectory data to correlated sequence of bus stops. Future works include integrating closed-loop incentive scheme into the system to valuate supply and demand, improving S2B algorithm so that K_{max} and K_{min} are adaptive, and introducing load balancing into the system for better scalability.

Acknowledgment

The authors would like to thank M.T. Maung, K.S. Ang, T.D. Tran, James Lim, and Bin He for their contributions in the system development. Support from A*STAR SERC Singapore grant is gratefully acknowledged.

References

[1] S. Solomon, G.-K. Plattner, R. Knutti, and P. Friedlingstein, "Irreversible climate change due to carbon dioxide emissions," *Proceedings of the National Academy of Sciences*, 2009.

[2] A. Thiagarajan, J. Biagioni, T. Gerlich, and J. Eriksson, "Cooperative transit tracking using smart-phones," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '10. New York, NY, USA: ACM, 2010, pp. 85–98.

[3] L. Richardson and S. Ruby, *RESTful Web Services*, 1st ed. California: O'Reilly Media, 2007.

[4] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. B. Srivastava, "Participatory sensing," in *ACM SenSys'06 World Sensor Web Workshop*, Boulder, Colorado, October 2006.

[5] E. A. Lee, "Cyber-Physical Systems - Are Computing Foundations Adequate?" in *Position Paper for NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap*, 2006. [Online]. Available: http://chess.eecs.berkeley.edu/pubs/329.html

[6] H. Zhuge, "Semantic linking through spaces for cyber-physical-socio intelligence: A methodology," *Artificial Intelligence*, vol. 175, no. 5-6, pp. 988 – 1019, 2011, special Review Issue.

[7] E. Agapie, G. Chen, D. Houston, E. Howard, J. Kim, M. Y. Mun, A. Mondschein, S. Reddy, R. Rosario, J. Ryder, A. Steiner, J. Burke, E. Estrin, M. Hansen, and M. Rahimi, "Seeing our signals: combining location traces and web-based models for personal discovery," in *Proceedings of the 9th workshop on Mobile computing systems and applications*, ser. HotMobile '08. New York, NY, USA: ACM, 2008, pp. 6–10.

[8] E. Lu, W.-C. Lee, and V. Tseng, "Mining fastest path from trajectories with multiple destinations in road networks," *Knowledge and Information Systems*, pp. 1–29.

[9] M. Kim, M.-O. Stehr, J. Kim, and S. Ha, "An application framework for loosely coupled networked cyber-physical systems," in *Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on*, dec. 2010, pp. 144 –153.

[10] R. K. Ganti, Y.-E. Tsai, and T. F. Abdelzaher, "Senseworld: Towards cyber-physical social networks," in *Proceedings of the 7th international conference on Information processing in sensor networks*, ser. IPSN '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 563–564.

[11] T. S. Dillon, H. Zhuge, C. Wu, J. Singh, and E. Chang, "Web-of-things framework for cyberphysical systems," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 9, pp. 905–923, 2011.

[12] A. Agarwal, M. Slee, and M. Kwiatkowski, "Thrift: Scalable cross-language services implementation," Facebook, Tech. Rep., April 2007. [Online]. Available: http://incubator.apache.org/thrift/static/thrift-20070401.pdf

[13] Land Transport Authority. (2009) PUBLIC TRANSPORT @ SG. [Online]. Available: http://publictransport.com.sg/publish/ptp/en.html

[14] National University of Singapore. (2009) Transport Services. [Online]. Available: http://www.nus.edu.sg/oed/services/transport/shuttle-bus-services.htm

[15] (2011) ContriSense:Bus - Android Market. [Online]. Available: https://market.android.com/details?id=com.contribus

[16] Fulmer, Jeffery. (2009) Joe Dog Software - Siege Home. [Online]. Available: http://www.joedog.org/index/siege-home