Enterprise Tracking Service Provider

Sanuri.D.Gunawardena[1], Lahiru.K.Kurukulasuriya[2], Charmy.Y.Weerakoon[3], D.D.Chamath M. Chandrasiri[4], Malitha N. Wijesundara[5]

Faculty of Computing, Sri Lanka Institute of Information Technology, New Kandy Road, Malabe, Sri Lanka

[1]sanudan91@gmail.com, [2]lahirukasun@gmail.com, [3]charmy.weerakoon@gmail.com, [4]chamathmc@live.com, [5]malitha.w@sliit.lk

Abstract — **Significant limitations exist in performance, security and reliability of object location tracking and remote sensing applications available today. Nevertheless, no researcher seems to have noticed this gap between existing conditions and desired conditions. Rather, researches are carried out to make such systems more cost-effective. This research focuses on improving the left out qualities which are critical for any similar real-time system, while keeping product cost down. The ultimate product is a complete, functional tracking service provider including server unit, client interface (both desktop and mobile) and mobile application to convert any mobile phone into a tracker. All these components are improved in performance, reliability, security and concurrency where applicable. In order to address this gap, technology bottlenecks are identified and are substituted with best suitable alternatives. The alternatives selection is done in two phases. First, the scope of selections is narrowed down based on sound evidence provided by previous studies done and information available from different sources. Secondly, a series of tests are carried out on sets of technologies and techniques selected for each component and the best suitable ones in each set are selected based on these test results. After implementing the system with the selections, a series of tests are carried out to determine the level of quality as a product.**

***Keywords*— Performance, Security, Reliability, Low development cost, Object tracking, Remote sensing**

I. INTRODUCTION

For the last few decades, applications of object location tracking and remote sensing concepts have served mankind in countless number of situations and researchers have worked hard on making these solutions cost-effective as they can be beneficially used for not only special purposes, but also in general daily activities. They range from day-to-day activities like tracking children, elders, disabled people, personal vehicles, pets, etc. to business situations like taxi companies, rent-a-car companies, wild life monitoring, fleet management, etc. and can be expanded further to special and critical tasks like tracking criminals, tracking suspicious activities through undetectable tracking devices, etc. Even though existing researches highly focus on cost-effective factor, no due attention is given to three other very important qualities of a real-time system, namely performance, security and reliability. This has resulted in highly inefficient usage of object tracking and remote sensing concepts today. Performance, security and reliability of a tracking service provider can be significantly improved by the careful selection of technologies and practices best suitable for each implementation related to the system and this research does the same. The final outcome is a system which is capable of providing services like remote data gathering of heterogenous objects, data storing, display of these data in a useful and understandable manner, allowing end users to manage the objects conveniently and perform all these tasks efficiently, securely and reliably. It not only provides location data of objects, but also other extractable information through sensors like temperature, pressure, speed, etc. given that sensors are capable of sending the data through SMS or GPRS to the server. This research proves that improvement of performance, reliability and security while keeping the development cost least is possible when correct technologies are chosen and correct practices are used. Also it presents those technologies and practices and their effect on the mentioned qualities. This is further demonstrated through the physical outcome of the research and its comparison with a leading, similar system called "OpenGTS"[1].

II. METHODOLOGY

Agile methodology was followed through out the research. The research was started with a traditional feasibility study which proved that it is financially, technically and operationally feasible because it is open-source technology-based, has no technological constraints and the final outcome was expected to cost least. Next, requirements of ETSP were gathered. Functionality expected from a tracking service provider were listed. Non-functional quality attributes involved were obvious from the research goals.

During design phase, structure of research product was visualized. ETSP consists of three major layers collaborating with each other to provide the user with a complete object tracking experience: server layer, tracker layer and client layer. Figure 1 depicts high level architecture of ETSP.

A. ETSP Architecture

1) Tracker Layer: Tracker layer represents all trackers that generate location and other sensor data and send them to the server. Trackers are of two types: Third party trackers of external vendors and mobile device trackers which are any mobile devices that have installed "ETSP mobile tracker". It is a mobile application developed by the research team as a part of this research, to collect sensor data from any mobile device

and send them to the ETSP server. This application is capable of identifying the sensors available in the device and sending retrievable data based on user configuration. It also supports both SMS and GPRS communication. User can determine how often to send data. "ETSP mobile tracker" is currently available for both Android and Windows phone 8 mobile platforms.

and take over in case of any main server failure. Database within the server stores all data involved with and generated by ETSP componetnts. User-end web application, SMS manager module that manages data received and sent through SMS, API is used to allow user-end mobile applications access tracking services, scheduler that sets appropriate scheduling algorithm and affinity for processes, configuration application that allows configuring ETSP for new customers,

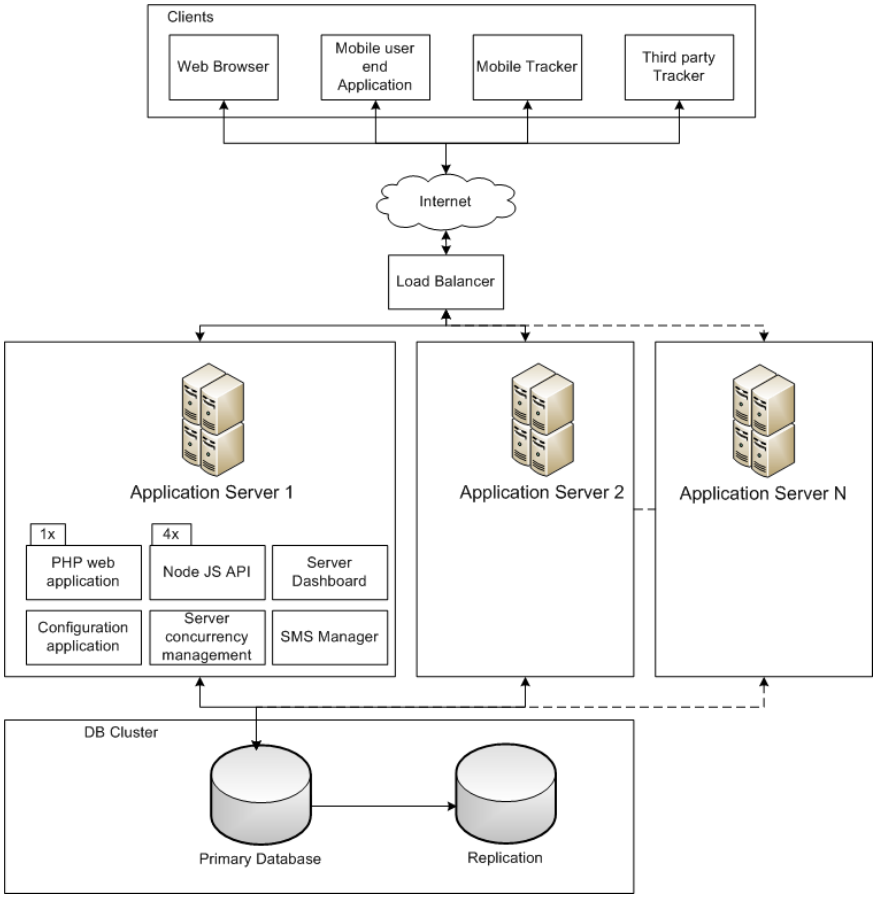

Fig. 1. High-level architecture diagram of ETSP

2) Client Layer: Client layer represents end-user's interface to ETSP services. This is where processed data is displayed to the users in an understandable form through graphical interfaces. This layer was completely developed by the research team as a web application and a mobile application version of the same. End-users are capable of managing user details, tracker details, object details, configuring third party trackers, tracking objects on maps and generating reports through this application. Mobile applications were developed for both Android and Windows phone 8 mobile platforms.

3) Server Layer: Server layer is the heart of ETSP. It bears the core functionality and all major processing of tracking service as well as the data store. Monitoring and backup server(s) identical to the main server is/are continuously synching with the main server to perform fail-safe operations monitoring dashboard that allows research team to monitor server status and related details and security module which performs security activities to prevent any unauthorized or malicious access to the server and database are the components that reside in server.

B. Implementation

Implementation of ETSP can be represented as a set of different phases and there will be many more to come as technology advances because this is a continuously improving research which has no defined termination. Up to the current status of the research, all activities performed can be bundled into following phases:

- Selection of technology alternatives for each module based on literature survey.
- Benchmarking of each alternative where necessary and comparison to others in the pool to select the best one.

- Actual implementation of ETSP and testing.
- Benchmarking of ETSP and comparison.

First and most important decision to be made was the server OS (Operating System). Several famous free and open-source OS were compared to select the best. Table I provides this comparison [2].

TABLE I
COMPARISON OF OS

Ubuntu	CentOS	Fedora
Most user-friendly Linux distribution	User-friendly	User-friendly
Offers strong community-based support	Offers support	Offers support
Has a consistent release cycle	Variable, longer release cycle	Variable release cycle
Reputation for security	High security	Moderate security
Not intended to be updated with all the latest software as time goes on. Designed to be stocked with long-tested software and only upgrading them with critical and security-related fixes	Software versions included with CentOS are rarely the latest	Continually update to newer software packages
Most stable	Most stable	Stable

Ubuntu 12.04 LTS was selected due to following reasons:

- This is a research project and the product is still in experimental stage.
- Only a limited time is available to complete the project. User-friendliness, strong community support, stability and security will ease up the development and those factors support the project goals of security, reliability, performance and scalability.
- Once the project is completed and developers can guarantee its functionality, product can be switched to a long-term supporting OS like CentOS easily.
- LTS stands for Long Term Support which is a version of Ubuntu that is fairly robust.

Next main concern was the database. Three very different, free and open-source Data Base Management Systems (DBMSs) were compared to choose the best suitable one. An abstract comparison is presented in Table II [3].

TABLE II
COMPARISON OF DBMS

MySQL	PostgreSQL	MongoDB
Relational DB	Object-Relational DB	Document store
A database schema is used.	A database schema is used.	No database schema is used.
Pure SQL	Pure SQL	NoSQL
No MapReduce.	No MapReduce.	MapReduce concept is available.
Uses foreign keys.	Uses foreign keys.	No foreign key concept.
Not big data solution.	Not big data solution.	Big data solution

MongoDB was selected as the best suitable database due to following reasons:

- ETSP involves a continuously growing huge amount of data which makes it a big data scenario where MongoDB serves well.
- Usage of MongoDB is extremely simple compared to options available in SQL DBMSs like primary keys, foreign keys, etc.
- No constraints are bestowed upon the database design as there are no schemas. Same collection have field-wise different documents.
- MongoDB is a promising technology with many powerful features like MapReduce, replica sets to provide high performance, security and reliability to data.

Database was planned and created. "Robomongo" was used by the research team as the interface to the database.

Tracker mobile application and user-end mobile application of ETSP were developed for two leading mobile OSs: Android and Windows phone 8. The special purpose of these applications apart from what they do is to expand the audience of users and increase overall system availability. In order to increase performance, best coding practices were used while developing these applications. For an example, switch-case programming constructor was used instead of if-else and "stringbuffer/stringbuilder" was used where string concatenation is required due to the known high performance of those options. Tracker data is compressed with LZ4 algorithm before sending to the server to make communication efficient. LZ4 was selected based on its compression ratio, performance and memory usage. This selection was done based on a previously done research [4]. Security was implemented through authentication and authorization. User-levels can be set through any combination of privileges available. Administrator can determine if each atomic functionality of the application is accessible by a certain user. Users can protect their application by applying a password. Data sent to the server are protected through encryption. Reliability is achieved by integrating the support for both GPRS and SMS communication methods into the application. These applications also include a set of rich features like alerts and notifications, geo-fencing, etc. Eclipse with adt-bundle and Visual Studio 2013 with WP SDK 8.0 were the tools used respectively to develop Android and Windows versions. Google and HERE maps were integrated to the two versions of user-end applications to enable mapping. Reports are generated by Node.js API and downloaded to the mobile device through the application

when requested. Mobile applications use REST communication with JSON strings to communicate with server through GPRS. JSON contributes to the performance of ETSP more than general "httpClient" and this was proved in benchmarking. Table III provides average benchmark results obtained.

TABLE IIIII
HTTPCLIENT VS. JSON BENCHMARK RESULTS

Operation	Avg. Time taken by HttpClient (ms)	Avg. Time taken by JSON (ms)
Get (1000 records of 240 bytes each)	678	461
Insert (50 bytes)	801	734
Update (74 bytes)	719	694
Delete (32 bytes)	310	283

Another important component of ETSP is user-end web application. PHP was chosen to implement this module due to its open source and free – base, easy Integration, flexibility and availability of resources. MVC architecture was used to make this application maintainable and expandable. Required performance, security and reliability were achieved through program code and application features in the same way as done in the mobile applications. This includes several extra features than mobile user-end application, like object path replay, statistics through fusion reports and customization of appearance of the application. Codeigniter was used as the development framework due to its high performance.

Configuration application and dashboard which reside in ETSP server were also implemented with PHP and Codeigniter based on the same reasons. Additionally, PhpSysInfo was used which allows to support new sensors in OS level which will allow monitoring of new aspects of server in future.

ETSP includes an API which allows mobile applications to access database. This API was first developed with PHP on Apache server to comply it with the user-end web application. The performance drawback of PHP was clearly noticed from the beginning. PHP was failing the performance target. Benchmarking proved this and the research team was forced to seek for another technology which is capable of providing real-time performance. A powerful and latest technology, "Node.js" was discovered at this stage. Benchmarking and comparing with PHP proved the significant improvement of performance provided by Node.js. The API was re-implemented in Node.js. Benchmarking results in average values are given in table IV. Security was also implemented at API level by allowing only registered i.e. authorized users to utilize API. Before allowing any database access, API checks whether the client or tracker is a registered, valid entity.

TABLE IV
PHP VS. NODE.JS BENCHMARK RESULTS

Operation	Avg. Time - PHP (ms)	Avg. Time - Node.js (ms)
Get (48 bytes)	0.49	0.3
Insert (240 bytes)	0.51	0.32
Update (240 bytes)	0.46	0.3
Delete (240 bytes)	0.47	0.15

While Node.js API handles GPRS data, a SMS manager was required to handle SMS traffic. This API reads SMSs received at GSM modem, authenticates and directs the message content to database. Also it forwards any messages like configuration messages to third party trackers. In order to create this, a free API called "Gammu" was used. C language was selected to build this module, based on the comparison of several famous and established languages, due to following reasons:

- C provides a performance more than any other language because it is compiled.
- C is a system-level programming tool.
- C has a better coordination with Linux kernel as kernel is also developed in C language. No system interfaces will be required for the programming and performance, security and reliability will be high.
- C is first programming languages in the world and hence has a large user community and support and a vast collection of libraries.

Yet, not only C language was considered. Bash shell scripting was benchmarked and compared to C because it is the native scripting language of Ubuntu OS. C was a clear winner compared to bash shell scripting. Table V provides benchmark results of these two languages. C shows an improvement of 5.25% of performance over bash shell scripting. So, SMS manager was implemented in C.

TABLE V
C VS. BASH SHELL SCRIPTING BENCHMARK RESULTS

Operation	Time taken by C (ms)	Time taken by Bash shell scripting (ms)
Print "Hello World" 1,000,000 times	30.7	32.4

Another critical concern of this research was the concurrency supported by ETSP. Concurrency goal was 1,000,000 concurrent connections per second at server. This is the combination of two communication forms: SMS and GPRS. Concurrency of these two should be handled separately at Node.js API and SMS manager.

Three solutions were identified to achieve concurrency of GPRS traffic through research survey:

- Server farm/Virtual Server farm
- Cloud computing (Infrastructure as a Service)
- Custom-modified kernel

Since server farms are highly expensive, research team first considered other two options. Cloud computing can be used, provided that vendor can provide GSM modems at server for SMS manager. Yet, this solution costs and fails performance target and is highly dependent on remote server's reliability as well. Then, the access to server reliability component of the

system is no longer with the research team. Next option was custom-modified kernel which was discovered as a result of literature survey done on million concurrency problem known as "C10M problem" [6]. This idea seemed ideal for this scenario but failed due to lack of knowledge, support and reliability because this concept also is still under research. "RumpKernel" is one such solution. Finally, research team tried increasing concurrency by high-level kernel tuning such as increasing number of file descriptors, reducing time wait of a request and increasing the maximum number of ephemeral ports and benchmarked ETSP. These tunings increased requests/second slightly. In order to achieve SMS client concurrency, SMS manager was developed such that any number of GSM modems can be connected to the server and used for the target purpose. Also, concurrency was increased by increasing the performance of SMS manager, thus reducing the time taken to process one circle of reading. This was achieved by setting CPU affinity and scheduling algorithm of the process. One-fourth of the CPUs available in the server are allocated for this purpose and process is executed according to FIFO algorithm.

Last concern of the research was to achieve the target reliability. Fail-safe operations were applied to Node.js API, Apache server, database and SMS manager. HAProxy was used as the load balancer because it is more reliable compared to hardware load balancers [7]. It balances the load into 2 or more Apache/Node.js servers. If several load balancers were used "Keepalived" tool can be used to manage failures among load balancers. It allows one load balancer to come online if other fails. Database replication was done using built-in MongoDB replica configuration. An arbiter is installed in the load balancer to assist in voting for database selection. Research assumes that several modems with SIMs bearing same contact number are used in SMS manager to ensure fail-safe. Dialog, Etisalat and Mobitel stated that this was not provided. In Sri Lanka, this is considered as a special scenario and a request has to be made to the Telecommunications Regulatory Commission of Sri Lanka.

Benchmarking of the system was done using Apache bench and testing scripts written by research team using node.js. Complexity and lack of support caused OpenGTS installation attempt to fail. Thus, reliable sources were used to obtain an idea about its quality levels.

III. RESEARCH FINDINGS, RESULTS & EVIDENCE

Final ETSP system was benchmarked for its performance, reliability, security and accuracy. The results are provided in this section. These results act as evidence which prove the current state of ETSP and that it has improved qualities than the competing product.

A. Performance

Most critical functionality of ETSP were considered for performance benchmarking: "Get" operation, "Insert" operation, "Update" operation, "Delete" operation. 1000 objects, 1000 trackers and 1000 location and other information records were used as the dataset. In order to determine performance level, response time at user-end and requests served per second were measured. Six of Intel core i5, 3.2 GHz, 8GB RAM machines were used for performance testing of ETSP with two machines hosting DB, another two hosting API and web application and other acted as the load balancer. Sixth machine was the client. Table VII provides ETSP performance summary.

TABLE VII
ETSP PERFORMANCE SUMMARY

Operation	Response time (ms)	Requests/Sec.
Get (24000 bytes)	0.28715	3485.9
Insert (240 bytes)	0.07273	13776.9
Update (240 bytes)	0.062125	16148.2
Delete (240 bytes)	0.054688	18369.8

Performance measurements for comparison of ETSP and OpenGTS were done with two core i5, 2.5 GHz, 4GB RAM machines. Values were measured for 1 instance of Node.js and based on minimum capability of running 4 instances on 4 cores of a single machine, maximum performance of ETSP on a single machine was estimated. Results showed that ETSP is ahead of two. Table VIII provides the results summary.

TABLE VIII
OPENGTS VS. ETSP PERFORMANCE SUMMARY

Operation	OpenGTS(1 instance)		ETSP(4 instances)	
	Response time (ms)	Requests /Sec.	Response time (ms)	Requests/Sec.
Get (24000 bytes)	29.1426	34.324	Cannot be estimated	636.32

B. Reliability

Mean time to recover was considered [1]. In the case of OpenGTS, if one server fails, end-user should manually change to another server. OpenGTS provides three such servers to access services and end-user should determine which one to use and configure OpenGTS accordingly before use. Hence, it is difficult to calculate reliability quantitatively for OpenGTS. ETSP is configured to automatically perform fail-safe operations within 1 ms.

C. Security

Security features of both ETSP and OpenGTS were assessed. Table IX provides this comparison. ETSP is clearly richer in security features than OpenGTS.

TABLE IX
SECURITY COMPARISON

Feature	**OpenGTS**	**ETSP**
Privileges	Only 2 user levels: • Admin: Configuration • Normal user: All tracking functionality	Customizable privileges • Each functionality can be allowed or restricted

Client connections	Only authorized users can access functionality.	Only authorized users can access functionality.
Tracker connections	Any mobile device can send tracker data.	Only registered trackers can send data.
Authentication	Authentication of, • users	Authentication of, • users • trackers
Encryption	No encryption	Encryption done
SQL injection	Can be done due to SQL based database	Cannot be done due to NoSQL based database

D. Accuracy of ETSP

In order to measure the accuracy of ETSP, tracking functionality was considered. This is the most important real-time operation of the system which is directly involved with accuracy. Mobile tracker was switched on and the time taken from first tracker data insertion to representation of object in user end mobile application was measured. High delay implies low accuracy as location will not be displayed accurately at any given time. Error was found by calculating the difference between actual value and measured value. Actual value was calculated by assuming that object moves with a constant velocity of $1ms^{-1}$. Error of ETSP was found out to be 0.0063256 meters while OpenGTS exhibited an error of 0.03011512 meters.

E. Unique Features of ETSP

Other than quality improvements achieved, ETSP is designed to provide many unique features:

- High scalability due to use of Node.js API
- Easy to setup and use
- Capable of managing heterogeneous objects
- Provides customizable reports
- Self-contained
- Enables alerts
- Supports geo-fences of custom shape, increasing accuracy

IV. CONCLUSIONS & FUTURE WORK

Unacceptable limitations in performance, security and reliability of modern object tracking applications violate the concept of real-time systems which they are supposed to be. This research is aimed at improving and thereby makes it to enterprise level with no degradation by finding best technologies to implement its modules.

Performance, security and reliability of the system entirely depends on the technology with which it is implemented. Using C language, high performance programs can be written. JSON can perform efficient client-server communication. Performance of an application can be improved by tuning the kernel level process management by setting affinity and scheduling algorithms. MongoDB is an excellent database management system to handle a massive amount of data. Applications can be made efficient, secure and reliable significantly by implementing best coding practices and concepts. With the combination of these technologies and practices, a significant performance improvement can be reached. Also, rich features included in ETSP which are discussed in this paper provide expected level of security and reliability.

As future work, user-space TCP/IP stack can be integrated to ETSP to achieve high concurrency with less resources. Research team also expects to implement the mobile applications in IOS platform with integration of richer features like statistics.

This paper provides the technologies and programming practices best suitable to achieve a certain level of performance, security and reliability in a real-time system. It is hoped that for any person who expects to build a similar system or any other real-time system, results of this research will be an aid and will provide insight on the performance, reliability and security level that can be expected with the combination of technologies considered in this paper.

ACKNOWLEDGMENT

First and foremost, we would like to thank the authority of Sri Lanka Institute of Information Technology (SLIIT) for providing us with a good environment and facilities to complete this project. Also, an honourable mention goes to our families and friends for their understandings and supports on us in completing this project.

REFERENCES

[1] M. Flynn, "OpenGTS - Open Source GPS Tracking System," O'Reilli, 19 May 2009. [Online]. Available: http://whereconf.com/where2009/public/schedule/detail/7086. [Accessed 20 August 2014].

[2] ThreeHosts.com, "Fedora Vs. Ubuntu Vs. CentOS - Server Reviews From ThreeHosts.com," PRWEB, 28 October 2013. [Online]. Available: http://www.prweb.com/releases/compare-linux-centos-vs/ubuntu-vs-fedora/prweb11273960.htm. [Accessed 22 August 2014].

[3] Greystoke1337, "Big Data," Wikipedia, 22 August 2014. [Online]. Available: http://en.wikipedia.org/wiki/Big_data. [Accessed 23 March 2014].

[4] I. P. Singh, "JMeter Load Testing Beginner Tutorial," 28 October 2013. [Online]. Available: https://www.youtube.com/watch?v=4mfFSrxpl0Y. [Accessed 1 September 2014].

[5] S. Karlsson and E. Hansson, "Lossless Message Compression," 2013. [Online]. Available: http://www.diva-portal.org/smash/get/diva2:647904/FULLTEXT01.pdf. [Accessed 14 April 2014].

[6] R. Graham, "C10M," Atom, February 2013. [Online]. Available: http://c10m.robertgraham.com/p/manifesto.html. [Accessed 22 August 2014].

[7] "HAProxy," [Online]. Available: http://www.haproxy.org/. [Accessed 25 June 2014].

[8] J. Harrell, "PayPal Engineering - Node.js at PayPal," PayPal, 22 November 2013. [Online]. Available: https://www.paypal-engineering.com/2013/11/22/node-js-at-paypal/. [Accessed 23 August 2014].