Scalable Processing of Geo-tagged Data in the Cloud

Martin Bauer, Dan Dobre, Nuno Santos, Mischa Schmidt

Abstract

The explosive growth of the mobile internet calls for scalable infrastructures capable of processing large amounts of mobile data with predictable response times. We have developed a scalable system supporting continuous geo-queries over geo-tagged data streams in the cloud. The experimental results confirm that our system scales, both in the number of supported geo-queries and throughput.

Keywords

Geo-fencing, geo-tagging, M2M, spatial index, stream processing

1. Introduction

Driven by the rapid growth of the mobile internet and the increasing adoption of Machine-to-Machine (M2M) technology, the amount of location data generated by location-enabled mobile devices is growing at an unprecedented rate. In the smart city context, automated M2M services have to deal with an increasing number of tracked objects, potentially with a high temporal resolution of the objects' location information (see **Fig. 1**). Thus, capturing, processing and acting upon location information in a timely manner demands for scalable infrastructure services. To address this problem, we developed a geo-fencing system for continuous queries over location streams in the cloud - a building block for programming smart cities.

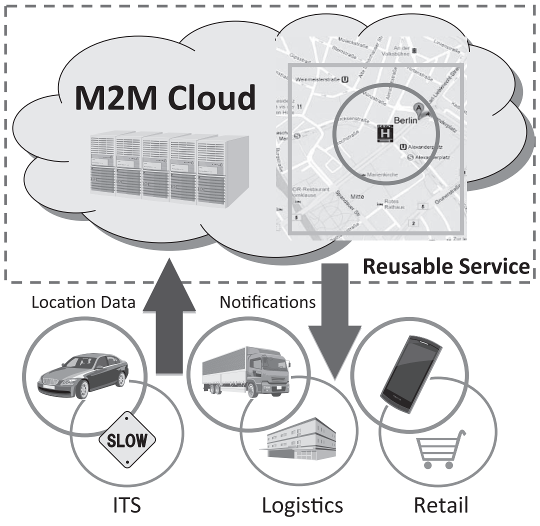

Fig. 1 Geo-fencing application areas.

The commonly observed industry trend is to share M2M infrastructure to realize gains in terms of infrastructure usage. Enabling functions such as geo-fencing are suitable for being implemented in infrastructure shared among different applications and application areas, further increasing the demands in terms of throughput and scalability.

Geo-fencing is a promising building block for the NEC M2M Platform Connexive ™ , improving competitiveness and opening avenues for new usage areas. Our work combines knowledge and technology from spatial databases, large-scale systems and distributed computing to attain the features of predictable scalable performance, timeliness and high availability. To the best of our knowledge no prior system simultaneously achieves all these features.

2. Objectives

We aim at addressing the following objectives:

(1) Scalability

One of our main goals is to scale according to the user demand. In the context of geo-fencing, scalability is required along two dimensions: (a) the amount of geo-fences set by users and (b) the load measured as the number of location updates per time unit. A scalable system provides both (a) low constant latency with an increasing number of geo-fences and (b) a throughput proportional to the number of machines.

(2) Predictable Performance

We regard predictable performance as a key property of any geo-fencing system. This translates to the requirement of a balanced system in the face of skewed distributions. Furthermore, popular locations must not result in hot spots.

(3) **Timeliness**
Locations should be immediately processed without the need of maintaining an index over them. However, quickly processing locations requires a suitable index for geo-fences.

(4) **High Availability**
The system should be available despite node failures. The aggregate capacity of the system should decrease proportionally with the fraction of unavailable machines.

3. System Model

Our system model consists of a set of mobile objects streaming location data in form of geographic coordinates to a cloud backend and a set of clients acting as subscribers for that data. The cloud backend processes the data and services client subscriptions. If a client wishes to track the location of a mobile object (e.g. a vehicle) relative to a geographic region of interest, it subscribes by registering a geo-fence. The cloud backend matches a stream of location data against the registered geo-fences, and continuously notifies the respective subscribers when objects of interest enter or leave the geographic region specified in the subscription.

4. Existing Approaches

The handling and processing of spatial data is not a new topic; likewise, the handling of streaming data at massive scale, a hot topic in cloud computing research. However, to the best of our knowledge, little effort has gone into bridging the gap between the handling and processing of spatial data and scalable, distributed architectures. This is where our work focuses on. In the following we briefly survey related work, explain their respective identified shortfalls, as well as the reasons why they, alone, are not suitable for fulfilling our goals.

4.1 Spatial Indexes

Spatial Indexes are specialized index structures oriented specifically towards efficient access to spatial data. We distinguish point data, represented as a pair of points, either Cartesian (X, Y) or GPS (Lat, Lon) coordinates, and region data, which are arbitrary polygons represented by a collection of pairs.

Binary-Space Partitioning trees such as Quad-, Oct-trees, KD-Trees as well as the recently proposed GeoHashing are well suited for point data, but do not work well with region data. In contrast, the R-Tree [1] and all its variants support both point and region data.

4.2 Spatial Databases

Several databases with support for spatial data exist, usually using an implementation of one of the aforementioned spatial structures.

Popular examples include MongoDB and PostgreSQL. MongoDB is a NoSQL storage system using GeoHashes for efficient access to point data, but without native support for region data. In contrast, PostgreSQL offers support for region data, but does not provide mechanisms for scaling beyond the capacity of a single node.

Despite their advantages, none of the existing mature spatial databases addresses the specific needs of a scalable geo-fencing system. Most related to our work is a recent paper describing how they built a spatial index on top of a CAN overlay [3] .

4.3 Streaming Systems

The tremendous growth of data generation and query rates have exposed many of the shortcomings of traditional database technologies, which are usually geared towards persistent and relatively static data. In contrast, stream processing systems address the needs of large-scale, real-time applications on highly-dynamic data, making some compromises on persistency.

Examples include Twitter Storm and Yahoo S4 [4] , both providing a scalable framework for processing data streams in real time.

Geo-fencing can expressed as executing continuous spatial queries on incoming location streams, which bears some similarity with streaming systems.

However, typical streaming systems apply simple operators to data streams. In geo-fencing, like in traditional databases, queries need to be persisted and indexed for efficient retrieval. Moreover, explicit control over replication and distribution schemes, which is highly desirable, is not natively supported in streaming systems. So a geo-fencing system needs to combine aspects of both stream processing and traditional database systems.

5. Our Approach

For ease of understanding, we distinguish between the basic approach and the scale-out mechanisms. We start by describing the approach in terms of computations carried out by a single node. Then we explain how we designed the system to scale in a data-center setting.

5.1 Basic Approach

At the level of single nodes, geo-fences are treated as region data made accessible over a spatial index structure (e.g. R-Tree). Location updates are treated as point queries to that region data. The spatial index stores only the Minimum Bounding Rectangles (MBRs), which is a rough approximation of the actual geo-fences. This has several advantages, one example being the compact size, which allows keeping the index in memory to speed up the search. A persistent key-value store (KVS) provides access to the geo-fence data.

For each point query, the spatial index is accessed to construct a candidate set of geo-fences whose MBRs contain the query point. In a second step, the actual geo-fences are retrieved from the KVS and the exact result set is determined via point-in-polygon.

In a straightforward approach, for each geo-fence in the result set, the corresponding subscribing client would be notified of an "inside" event. Likewise, for each geo-fence not contained in the result set, the corresponding subscriber would be notified of an "outside" event. Each subscriber would then locally determine the events corresponding to an object actually crossing a geo-fence boundary. Obviously, this approach is naive because notifications are sent to every subscriber even if the boundary of her geo-fence(s) has not been crossed.

Rather than notifying subscribers about the current device location relative to a geo-fence, the system should notify only about "enter" and "leave" transitions. For this purpose, knowledge about the previous location of an object is required (see **Fig. 2**).

Given two point queries p and p' corresponding to the previous and the current location of an object and the corresponding result sets R and R', the "enter" and "leave" transitions E and L can be simply computed as follows:

(1) $E = R' - (R \cap R')$

(2) $L = R - (R \cap R')$

Subsequently, only the subscribers corresponding to the set E (resp. L) are notified of an "enter" (resp. "leave") event (see the illustration in **Fig. 3**).

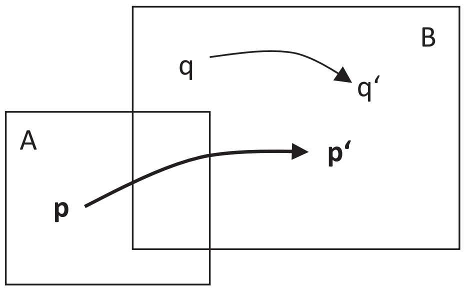

Fig. 2 Subscribers A and B should be notified only of the transition p → p'.

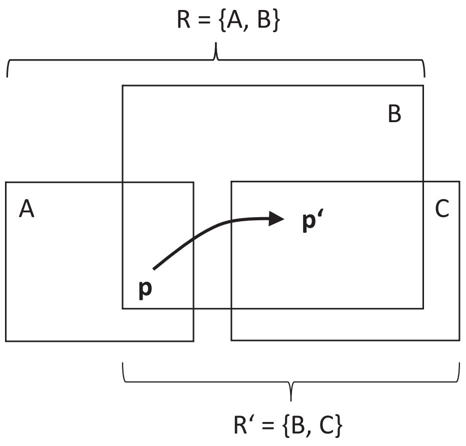

Fig. 3 According to equations (1) and (2) E = {C} and L = {A} and thus subscriber B is not getting unnecessary notifications.

5.2 Scaling in the Cloud

Now we explain how we designed the system to scale in a cloud setting.

Recall that there are two dimensions in which a geo-fencing system needs to scale: in terms of (a) the number of geo-fences and (b) the throughput expressed as the number of locations processed per time unit.

To accommodate both, our approach organizes the machines in a logical grid, inspired from the work on grid quorum systems for read/write storage [2)]. In a nutshell, a quorum system is a set of pairwise intersecting sets of nodes. The intersection property guarantees that a read operation overlaps at least one node modified by a write. A grid quorum system is a

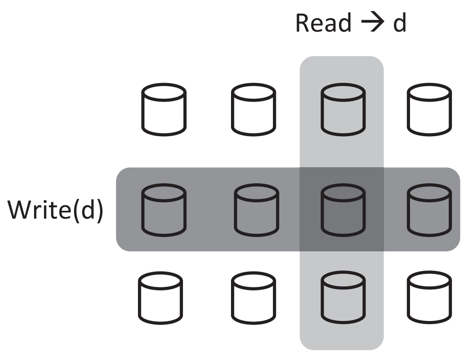

Fig. 4 Grid quorum system where each data item is written to a single row and read from a single column.

special type of quorum system, in which the set of nodes are organized in rows and columns (see **Fig. 4**). If a data item is replicated to an entire row and a read hits an entire column, then the read receives a reply from one node updated by the write. Consequently, full coverage is ensured.

The advantage of grid quorum systems is that throughput can be easily scaled by increasing the number of rows and/or columns.

Translated to our problem, a client setting a geo-fence corresponds to a write operation, while locations by devices correspond to read operations. More specifically, when a geo-fence G is set, G is disseminated to an entire row (see Fig. 4). Since a row is essentially a set of replicas, rows are called clusters. Each node in the cluster stores G and indexes its MBR in the way described in the previous section. Location updates are disseminated to at least one node from each cluster. Each of these nodes locally computes the sets L and E and notifies the corresponding subscribers. Coverage of G is guaranteed by the quorum intersection property (the dark shaded area in Fig. 4).

We anticipate the geo-fencing workload to be mainly read-dominated, which translates to scaling the throughput in terms of location updates rather than operations on geo-fences. Also the system should be able to absorb a sudden increase in load. This requires quickly adding new replicas to a cluster. To be able to do so, clusters are maintained via group membership (GM), enabling to increase capacity by adding new replicas via the join() and state_transfer() functions provided by a GM middleware. Other examples of required functionality typically offered by a GM subsystem are failure detection and replica eviction.

Besides throughput scalability, we need to accommodate an increasing amount of geo-fences without swamping individual nodes. Thus, we should be able to add new clusters, and do so without the need for reshuffling the entire data set. To tackle this problem, we use consistent hashing to map geo-fences to clusters of nodes. Consistent hashing ensures that under additions of new clusters (a) every cluster stores roughly the same amount of data items and (b) only new clusters receive data items. For instance, with C clusters (after addition of one cluster), no more than 1/C of the data items need to be migrated to the new cluster.

5.3 Evaluation

Our approach has been experimentally evaluated on a small cloud based on OpenStack virtualization. The results,

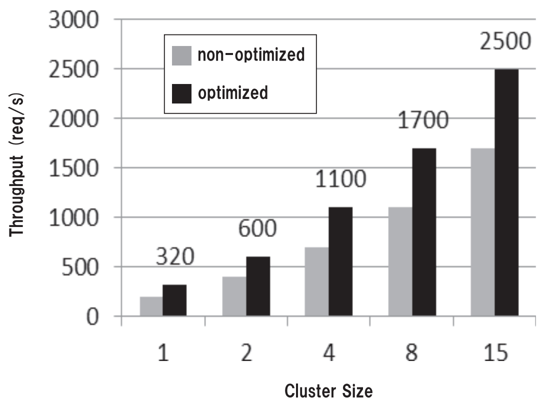

Fig. 5 Throughput vs. cluster size (5Million geo-fences). Y-Axis: Throughput (locations / sec), X-Axis: Cluster size.

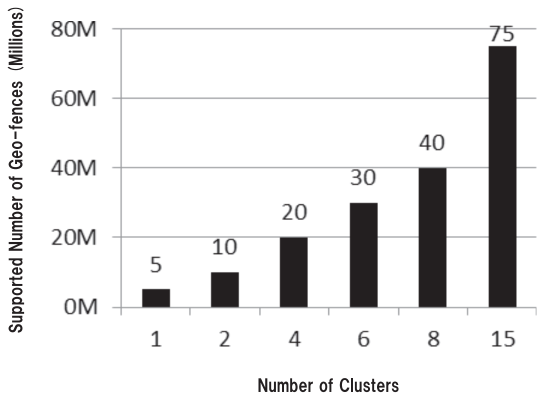

Fig. 6 Data scalability with constant low latency (to 20ms). Y-Axis: supported number of geo-fences (Millions), X-Axis: number of clusters.

measured on two desktop PCs with 8 cores each (total cost lower than 1600$) demonstrate the scalability and highlight the performance/price ratio. The throughput, measured as the number of locations matched against a large number of geo-fences, scales almost linearly with the cluster size (see **Fig. 5**), while latency remains constant despite an increasing number of geo-fences (see **Fig. 6**). Fig. 5 also shows scalability optimizations we realized by proactively caching results which we will describe in a future publication.

6. Conclusion

We presented a scalable system for processing geo-tagged data in the cloud. At the core is a shared-nothing architecture that scales proportionally with the number of nodes, avoiding bottlenecks and single points of failure. To obtain a uniform distribution, geo-fences are mapped to nodes irrespective of their spatial attributes (possibly highly skewed data), keeping the system balanced even under worst-case distributions. The downside of this approach is that location updates need to be matched against the entire dataset, penalizing throughput. Taking advantage of spatial partitioning to improve throughput, while keeping the distribution of geo-fences uniform under skew, is subject of future work.

References

1) A. Guttman: “R-trees: a dynamic index structure for spatial searching,” SIGMOD ’84 Proceedings of the 1984 ACM SIGMOD International Conference on Management of data, pp.47-57

2) S.Y. Cheung et al.: “The grid protocol: A high performance scheme for maintaining replicated data,” ICDE 1990, pp.438-445

3) J. Wang et al.: “Indexing multi-dimensional data in a cloud system,” SIGMOD ’10 Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, pp.591-602

4) L. Neumeyer et al.: “S4: Distributed Stream Computing Platform,” Data Mining Workshops (ICDMW), 2010 IEEE International Conference on.
http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5691154

Authors' Profiles

Martin Bauer
Senior Researcher
NEC Laboratories Europe
NEC Europe Ltd.

Dan Dobre
Research Scientist
NEC Laboratories Europe
NEC Europe Ltd.

Nuno Santos
Research Associate
NEC Laboratories Europe
NEC Europe Ltd.

Mischa Schmidt
Senior Researcher
NEC Laboratories Europe
NEC Europe Ltd.

Vol.7 No.2 Big Data

Vol.7 No.2

September, 2012

Special Issue TOP