

Scalable Architecture for Context-Aware Activity-Detecting Mobile Recommendation Systems

Michael Roberts, Nicolas Ducheneaut, Bo Begole, Kurt Partridge, Bob Price, Victoria Bellotti,
Alan Walendowski, Paul Rasmussen

*Palo Alto Research Center
3333 Coyote Hill Rd, Palo Alto, CA 94304
mroberts@parc.com*

Abstract

One of the main challenges in building multi-user mobile information systems for real-world deployment lies in the development of scalable systems. Recent work on scaling infrastructure for conventional web services using distributed approaches can be applied to the mobile space, but limitations inherent to mobile devices (computational power, battery life) and their communication infrastructure (availability and quality of network connectivity) challenge system designers to carefully design and optimize their software architectures. Additionally, notions of mobility and position in space, unique to mobile systems, provide interesting directions for the segmentation and scalability of mobile information systems. In this paper we describe the implementation of a mobile recommender system for leisure activities, codenamed Magitti, which was built for commercial deployment under stringent scalability requirements. We present concrete solutions addressing these scalability challenges, with the goal of informing the design of future mobile multi-user systems.

1. Introduction

In a world of information overload [1], finding the right content to consume has become increasingly difficult. Recommender systems were designed to address this issue by filtering information down to a restricted set of items that would most likely satisfy their users [2]. They have become extremely popular of late and are now used routinely to recommend diverse items to consumers on the Internet [3]. But until recently these systems were limited to applications accessed from a single machine, with the development of mobile recommender systems still in its infancy [4].

Depending on the algorithm used [3] recommender systems can be highly computational, which adds to the challenge of migrating them to mobile devices. It is perhaps unsurprising, therefore, that most deployments of mobile recommender systems in a research context have been limited to small user populations, ranging from 10 to 60 total users [e.g. 7,8,9,14]. And while the research community has discussed recommender scalability in the past [e.g. 13], it has remained focused on non-mobile usage. Consequently, the issue of how to design a high-performance mobile recommender system, scalable to hundreds, if not millions, of simultaneous users, remains to be addressed.

The design of scalable systems for web applications has received much attention recently [11, 12, 15], with approaches concentrating on fault tolerance, distribution of load, storage of large datasets, providing context for front-end operations and other similar needs. Scalability metrics can include density of users per server instance, concurrent user population, and response time.

We approached the problem of creating a scalable mobile recommender system directly during the development of Magitti. The sponsor of this research, Dai Nippon Printing Co., Ltd., had set an ambitious scalability target for the first prototype: it had to support at least 400 simultaneous users ranking 3000 points of interest on a single commodity server. To meet this requirement we used techniques that are broadly applicable to mobile recommender systems. Our goal with this paper is to describe these techniques, thereby directly addressing the research gap mentioned above.

We begin with a presentation of the overall architecture of Magitti, paying particular attention to how we streamlined data flows between client and server through the use of spatial structures and caching

data stores. We then explore how we optimized the recommendation process by using a stateless, ad-hoc chain of content-scoring models. We briefly describe the custom testing infrastructure we developed and how it helped us reach our performance targets. Finally, we conclude with a discussion of the lessons learned from this effort.

2. Architecture and Implementation

In general, the implementation of Magitti follows a conventional client server approach, with most significant operations (other than UI) performed on the server. A conceptual overview of the general approach for the system is provided in Figure 1. For the client, running on a Mio A701, we wrote a custom application in C# to provide a rich user-interface experience (see [18] for details of the interface design). Clients connect to the server system using a WAN connection providing TCP-IP access across T-Mobile's Internet service, which allows the application to communicate over http to an Apache Tomcat-based server framework. The Mio A701 has an embedded GPS receiver, which provides location context for recommendation calculations. For sending messages to the server, we used simple post strings, with returning messages encoded using JSON [16]. The use of JSON allowed us to compose and decode complex messages with only a small amount of overhead.

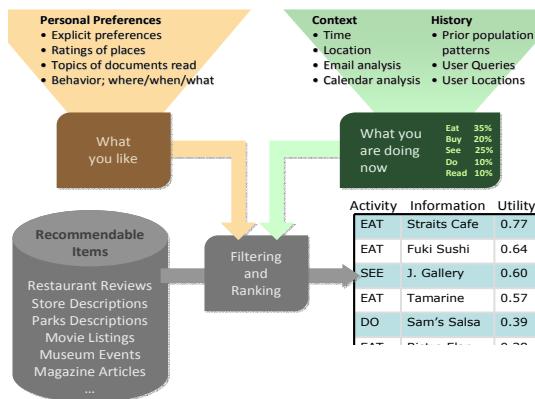


Figure 1 – Conceptual architecture

In general, we avoided performing large amounts of computation on the mobile device, with the aim of reducing the device's power consumption. We found that enabling the GPS device alone had a significant effect – with GPS enabled our devices had only a 3-4 hour battery life, as opposed to approximately 7 hours without it. This made it necessary to limit additional

draws on power. Use of server-side computation resulted in response times comparable to other mobile web applications, with the majority of the latency coming from the WAN network; under ideal local network conditions, our server system is capable of providing a complete recommendation response in less than 50ms.

Our major objective was to perform complex recommendation calculations while supporting our scalability targets. Each time the device's context changes (whether through position change, time change, or user input), the system computes a new set of recommendations based on the new context and the user's personal tastes. The server computes the utility to the user of a number of "recommendables", which are items of content describing a venue or event, such as stores, restaurants, parks, art galleries, museums, beauty salons, movie listings, etc. Each recommendable has information about the item's location, hours of operation, text description, and venue-specific pieces of information such as (for restaurants) price range, cuisine, smoking, alcohol served, etc. Recommendables also have a set of ratings that users of the system have assigned to them, along with the average rating.

The first step to minimize the time needed for utility calculations was to put all of the data needed into an in-memory data store so that slow and potentially non-scalable database access was not frequently required. Once this caching layer was deployed, the calculation of the recommendations became the focus for optimization. In addition to (1) caching, the two ways to minimize the time for calculations were to (2) avoid calculations unlikely to result in changes to our result sets and (3) optimize the computations when unavoidable, which took the form of both local computation optimization and also movement of expensive operations into offline processing pieces. Many of the expensive operations were concerned with activity modeling, in which we analyze the user's movement and query patterns, and with collaborative filtering, in which similarities between user's tastes are analyzed to provide enhanced recommendations. The following sections elaborate on optimizations used.

2.1. Caching and Database Structure

Database access in Magitti is structured via a notion of repositories. Each repository buffers and caches a particular type of data from the database: (1) read-only repositories that proxy data read in at application start-

up time, and (2) repositories that accumulate state for periodic writes. The second type of repository is typically used to store data for offline processing, such as performing user clustering for collaborative filtering and activity modeling, which are too expensive to perform in real-time in a server instance. The algorithms used are described in detail elsewhere [17].

All repositories serve the same process: to insulate the main application layer from having to access the database directly, and to provide a high level API for data-storage via key-value pairs. Note that our approach to database access could eventually be replaced with a more optimized key-value storage system, similar to Amazon's Dynamo [12], without disturbing higher level application functionality.

When data in the database is to be updated, we provide for asynchronous updates wherever possible via a write-through cache. A command pattern [20] is utilized for data insertion. Instead of the calling application thread writing directly to the database, commands containing updates are placed into a queue, and executed using a thread-pool, which parallelizes database access by allowing many simultaneous database read/write operations to be in progress. Any I/O blocking then occurs in the database update threads, as opposed to the main utility calculation thread.

2.2. Spatialized data storage

We developed a spatial data store that caches and partitions our content database along geographic bounds. Our use of the spatial structure ensures that calculations are performed only on recommendables that are within a specified geographic region, avoiding comparative calculations over the entire repository. Spatial structures are well known in other domains, such as computer graphics [21].

The structure used is based on grids of connected cells, each of which covers a particular geographic area (typically a 200m tile). Extent of the grids is determined by an initial pass through all recommendables, examining their latitude and longitude co-ordinates. At server start up time, we create grid cells and sort recommendables by geographic location (latitude and longitude) using simple numeric operations to determine which cell they reside in. For querying, we perform a similar process, first spatializing the query co-ordinates to determine the cell in which the co-ordinates reside then returning

lists of cells around the computed cell. Since the cell retrieval process is entirely based upon simple numeric scaling operations, it runs in constant time, regardless of the number of items in the structure or the size of the grid. Using this mechanism, we can radically cut down on the number of calculations which need to be performed by performing calculations only on recommendables close to a given location, and not on recommendables further away.

The spatial partitioning model has some potential downsides, one of which is that a specified geographic boundary may contain too few items to recommend to the user. To alleviate this problem, we implemented an automatic expansion of the query range when the result set contains only a few items. This addressed another interesting aspect of location-based venue recommendations which was that each type of recommendable venue has different geographic density, both over all regions as well as within specific parts of a city. For example, there are generally more stores in a given area than there are parks. Likewise, there are generally more stores than restaurants per unit square in a shopping center, while the inverse is often true of a town/city center. Another problem concerns the risk that the system might miss items just outside the bounded range that would have a high utility due to a high match with the user's personal preferences. For the Magitti prototype, we decided to take the risk that high-scoring items might be missed and evaluate whether users would identify this as a significant problem. In fact, our user evaluation found that most users wanted even tighter bounding than we had designed for [19].

2.3. Recommendation engine

Most recommender systems in commercial use today implement a form of collaborative filtering (CF) [2]. Intuitively, this technique tries to automate propagation of information among like-minded people. Using ratings assigned to items in the past, CF recommenders compute "neighborhoods" of users with similar tastes (often using statistical correlation measures). Unrated items for a user are given a score based on the ratings assigned to the same item by other users in the neighborhood (the prediction can be as simple as a mean rating, but more complex techniques are also available) [5]. However, CF-only systems neglect contextual information that could be crucial in a mobile context [6]. Past research has also shown that the accuracy and usefulness of predictions often benefit

from a hybrid approach combining CF with other techniques [10].

We therefore decided to adopt a mixed-model, ad-hoc architecture, for Magitti’s recommendation engine. Our system maintains a library of “utility models” tasked with computing the utility (or score) of an item. Models contribute to an item’s final score only if the information they need is available. This lets us combine contextual data with more static preferences to maximize the usefulness of our predictions to the user: for instance, our Distance model will recommend closer places if the weather turns bad, but it can also function perfectly well without this contextual data by defaulting to an exponential decay utility function based on a user’s range. Similarly, a user having rated a lot of items will trigger the inclusion of a CF-based model in the utility computation chain, but another user without any rating (and consequently with CF disabled) will still receive recommendations based on the integrated score from all the other models available. A Set Generator is responsible for combining the models using rules that can be hard coded or learned over time (we discuss this part of our system in more depth in [17]). In Magitti’s current implementation we combine up to eight models.

However, it is obvious that a consequence of the hybrid approach we adopted is a significant increase in computation costs. This forced us to pay particular attention to the kinds of data structures used in each model, avoiding expensive operations, and spending time doing profiler informed optimization.

For memory optimization, we needed to minimize the number of objects stored as an item moves through the model chain. Our models are stateless: after computing an item’s utility they move on to the next item without storing the result, keeping track of the utility computed by each model only for the current item. When the final utility is obtained, the Set Generator uses a binary tree limited to the number of recommendations the user wishes to see (typically 20) to store an Item ID/utility pair, and this only if the current item’s utility is better than those of the items currently in the tree.

2.4. Learning and Offline Modeling

As mentioned earlier, some context for recommendations is provided by the handheld device. Additionally, we use information obtained from user studies [18] to provide probabilities (“priors”) for user’s activities varying by day of the week, time of

day, weather, and neighborhood. However, as users move through the world performing queries and examining recommendables on their handheld devices, it is possible to build a more detailed model of their behavior. We use two primary data sources: venues close to reported GPS positions, and records of how searches are refined on the client device. These data sources are indexed by minute of week, averaged with other data near the same minute of week, and then used to improve activity predictions and recommendation preferences by learning weights for our utility models (we refer to this process as model-weight learning). For example, if a user visits and searches for fast food restaurants for lunch, but pricier ethnic restaurants for dinner, the learning would identify this pattern and apply it to new recommendations.

Some of the algorithms used in the modeling process are computationally intensive. To maximize the density of users per server instance it is not desirable to run these applications in the same process as the server, since they typically do not run in real time. In particular, we perform offline modeling for parts of the system performing interference about activities via GPS trace analysis, preferences for reading particular types of content, etc. The offline process is expected to be run about once per day; however as it need not be executed on the web server, it could be run continuously in the background. When the user logs in, the smaller digest files and information in the database generated by the offline processes are read into memory and cached there for the duration of the session. Larger per-user sets of information are loaded in lazily to reduce the per-user memory footprint, which is also reduced using other compression techniques; primarily the compression of objects which utilize only a small part of a number type’s range into a corresponding smaller number type object.

2.5. Testing, tools, and performance evaluation

Because we wanted to be able to examine the behavior of the system before we had real-world users, we needed a way to simulate both the motion of users in the world and having large numbers of users connected to the system. Additionally, the complexity of the approach applied to the generation of the recommendations required that we be able to visualize the various properties of the server’s internal models. Thus, to aid the development process, we created a suite of testing applications – a simulator (which enabled us to simulate the motion of users in the world), a visualizer (which enabled us to examine the

server state), and a re-player application (which enabled us to replay the motion of users in the world). All three of these applications communicate with the server via a shared multi-threaded network interface. Using a task queue, any of the applications can place messages into the network interface, which will then asynchronously contact the server and perform various types of operation.

The simulator allowed us to place a realistic load of up to 500 users into a single server instance and proved invaluable for testing. With the simulated load in place, we were able to profile the runtime execution of the server and optimize code where necessary. For a typical load of 400 users, the final system exhibited (local) query times of approximately 50ms. At loads of more than 500 users per server instance, we started to become limited by network IO processing overhead, with corresponding degradation in response time.

4. Future work and conclusions

We have demonstrated both that a client-server based mobile recommendation system is practical for large-scale deployment and that a good density of users per server node can be obtained. With the current system, we do the majority of the computation on the back-end server infrastructure. With increasing capability of mobile devices, it should be possible to perform more of this work on the client; in particular, systems which allow for the transparent migration of data and computation between small devices and the more extensive facilities provided by larger (server) devices are interesting directions to pursue. However, particularly in the case of algorithms like collaborative filtering, it still makes sense to have extensive server build-out.

For deployment at commercial scale, we envision clusters of self-similar servers behind commercial high availability load balancing solutions. Clusters can be geographically segmented, with a separate login step in which a device provides a geographic context to the login server, and is then directed to a particular cluster for the duration of its session. For a comprehensive approach dealing with failures of live servers, a distributed key value store similar to [12] can be used to store session data, thus enabling the same context to be served by different server instances inside a cluster.

We learned a number of lessons during the project; in particular, our early development of simulation and visualization tools allowed us to both place a realistic

load on our server architecture, and examine the internal state of the server system easily. Caching and spatial partitioning mechanisms served a vital role to insulate ourselves from database access and also provide a layer capable of interacting with more advanced storage mechanisms in the future. Likewise, our use of standard web technology in a mobile setting worked well. We believe similar approaches could be easily applied to future mobile projects.

10. References

- [1] Lyman, P., Varian, H.R., "How Much Information?", <http://www.sims.berkeley.edu/how-much-info> (2003).
- [2] Resnick, P., Varian, H.R., "Recommender systems", *Communications of the ACM*, ACM, New York, 1992, 40 (3), pp. 56-58.
- [3] Schafer, J.B., Konstan, J., Riedl, J., "Recommender systems in e-commerce", *Proceedings of EC99*, ACM, New York, 1999, pp. 158-166.
- [4] Miller, B., Albert, I., Lam, S.K., Konstan, J., Riedl, J., "MovieLens Unplugged: Experiences with a Recommender System on Four Mobile Devices", *Proceedings of HCI 2003*, British HCI Group, September 2003.
- [5] Adomavicius, G., Tuzhilin, A., "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions", *IEEE Transactions on Knowledge and Data Engineering*, IEEE, New York, 2005, 17 (6), pp. 734-749.
- [6] Adomavicius, G., Sankaranarayanan, S.S., Tuzhilin, A., "Incorporating contextual information in recommender systems using a multidimensional approach", *ACM Transactions on Information Systems*, ACM, New York, 2005, 23 (1).
- [7] Cheverst, K., Davies, N., Mitchell, K., Friday, A., Efstratiou, C., "Developing a context-aware electronic tourist guide: some issues and experiences", *Proceedings of CHI 2000*, ACM, New York, 2000, pp. 17-24.
- [8] Poslad, S., Laamanen, H., Malaka, R., Nick, A., Buckle, P., Zipf, A., "CRUMPET: Creation of user-friendly mobile services personalized for tourism.", *Proceedings of 3G*, London, UK, 2001, pp. 26-28.
- [9] Ricci, F., Nguyen, Q.N., "Acquiring and revising preferences in a critique-based mobile recommender system", *IEEE Intelligent Systems*, IEEE, New York, 2007, 22 (3), pp. 22-29.
- [10] Burke, R., "Hybrid recommender systems: survey and experiments", *User Modeling and User-Adapted Interaction*, Kluwer Academic Publishers, 2002, 12 (4), pp. 331-370.

- [11] "Seattle Conference on Scalability", http://www.google.com/events/scalability_seattle/ (2007).
- [12] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G. Lakshman, A, Pilchin, A, Sivasubramanian, S., Vosshall, P., and Vogels, W., "Dynamo: Amazon's Highly Available Key-Value Store", *Proceedings of the 21st ACM Symposium on Operating Systems Principles*, ACM, New York, 2007, pp. 205-220.
- [13] Sarwar, B.M., Karypis, G., Konstan, J., Riedl, J., "Recommender Systems for Large-Scale E-Commerce: Scalable Neighborhood Formation Using Clustering", *Proceedings of the Fifth International Conference on Computer and Information Technology (ICCIT 2002)*, December 2002, East West University, Bangladesh.
- [14] Tung, H.-W., Soo, V.-W., "A Personalized Restaurant Recommender Agent for Mobile E-Service.", *Proceedings of EEE 2004*, IEEE Computer Society, Washington, DC, 2004, pp. 259-262.
- [15] Chang, F., Dean, J., Ghemawat, S., Hsieh, W., Wallach, D. Bigtable, "A Distributed Storage System for Structured Data", *Proceedings OSDI 2006*, USENIX Association, Seattle, WA, November 6-8, pp. 205-218.
- [16] Various, *Introduction to JSON*, <http://www.json.org>.
- [17] Ducheneaut, N., Huang, Q., Partridge, K, Price, B, Roberts, M., Chi, E., Bellotti, V., Begole, B., "Collaborative Filtering is not enough? Experiments with a mixed-model Recommender for Leisure Activities", *PARC Technical Report*, Palo Alto Research Center, 2008.
- [18] Belotti, V., Begole, J., Chi, E., Ducheneaut, N., Fang, J., Isaacs, E., King, T., Newman, M., Partridge, K., Price, B., Rasmussen, P., Roberts, M., Schiano, D., Walendowski, A., "Activity Based Serendipitous Recommendations with the Magitti Mobile Leisure Guide", *CHI 2008 Proceedings*, ACM, , Florence, Italy, pp. 1-10.
- [19] Isaacs, E., Belotti, V., Walendowski, A., Newman, M., Ducheneaut, N., "The User Experience of a Context-Aware Personalized Mobile Recommender of Leisure Activities", *PARC Technical Report*, Palo Alto Research Center, 2007.
- [20] Gamma, E., Helm, R., Johnson, R., Vlissides, J., *J. Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995.
- [21] Langetepe, E., Zachmann, G., *Geometric Structures for Computer Graphics*, AK Peters, Location, 2006.