Why Client-based Dynamic Caching for Map Rendering

-Comparison with Google Map's tiling approach

Ahmet Sayar - 12/05/07

This document refers to the caching approach mentioned in [3]. Summary of approach: We allocate separate chunk of caching area for the clients and each client is served from its own allocated area. Client's allocated area is updated with the data used for serving that client's last query. Server differentiates the clients based on their IDs defined in the request. More details are given in [3].

Motivation to develop such a model: In short, we develop this framework purely for performance reasons. It removes the repeated jobs and helps efficient load balancing over the un-predicted workload by utilizing the locality [1] and nearest neighborhood [2] principles. Locality principle in this context is explained as following. If a region has a high volume of data, then the regions in close neighborhood also expected to have high volume of data. The simplest example to give is the distribution of human population data across the earth. The urban areas have higher human population than the rural areas, and furthermore oceans (2/3 of the world) have no human populations etc.

Architectural details and test results (in [3]) are particularly given for GIS domain but principles can be applied to general domain. Here I explain why and when such an algorithm is needed by comparing it with Google Map Server's tiling approach in map rendering.

The fundamental concept behind the caching is removing the resource consuming repeated jobs and serving the client from the ready to use data sets kept in local storages. In case of map rendering process, ready to use data sets are map images.

Google Map Servers are the best examples for caching map images to provide high performance map services. They keep the data as ready to use map images chunked in tiles. Each tile is defined by its x,y coordinates and a corresponding zoom-level (18 different zoom levels). The major thing they do is formalizing the accepted requests (in terms of parameters), and responses in terms of the tile compositions. Their major concern is developing high-performance map services. In order to do that, they introduced AJAX (Asynchronous JavaScript and XML) for client/server communications and used locally stored static map images.

However, in case of considering (1) data's dynamic and distributed characteristics, and their various heterogeneous formats; and (2) seamless addition of new data sources rendered as layers and overlaid with other layers in various combinations and orders, Google Map's static caching approach (tiling) would not work.

Google Map Servers provide two unique layers, satellite and Google map, and one hybrid layer as overlay of those two. Maps are served from three groups of tiles corresponding to these

layer sets. In order to highlight the limitations of their algorithms, let's assume they provide three unique layers instead of two. Let's say layer names are 'a, b and c. Then server would need to have 7 different tile groups as named a, b, c, ab, ac, bc and abc.

In summary, for the N number of unique layers, the required number of tile groups is calculated as below. It is sum of all k-subset combinations in which k gets the values from 1 to N.

$$\sum_{k=1}^{N} \mathrm{C}\binom{N}{k} \quad , \qquad \mathrm{C}\binom{N}{k} = \frac{N!}{(N-k)!\,k!} \tag{1}$$

In case of 10 unique layers (N=10), the number of tile groups would be 1023. Moreover, in each tile group there are thousands of tiles, and each tile in the group has different copies for 18 different zoom levels. As the layer number increases, the number of required tile groups increases dramatically and at some point it becomes impossible to store that much tiles in a single storage with current possible technologies.

Client-based dynamic caching approach: We allow all the data to be kept at their original resources and integrated to the system through standard service API, communication messages and in expected common data formats. This enables extensibility and interoperability, easy data handling/maintenance, and workload and data sharing. We do on-demand data fetching and rendering. Instead of caching whole combinations of data sets we fetch and cache the data based on client's actions (locality and nearest neighborhood principles). Clients are given the flexibility to compose their own maps based on their applications' requirements. The framework also enables attribute based querying of the data integrated to the system through the common data model carrying both content and presentation features of the data.

With the client-based caching, besides removing the repeated processing jobs, we utilize the locality principles and develop efficient load balancing algorithm for sharing unpredicted workload among the worker nodes.

In short, even though both systems provide map images as final outcome, their concerns and focus are different in terms of architectural principles and performance design features.

REFERENCES

[1] P.J. Denning and S. C. Schwartz, Properties of the working-set model. Communications of the ACM, 15(3), March 1972.

[2] Belur V. Dasarathy, editor (1991) *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, ISBN 0-8186-8930-7.

[3] Ahmet Sayar, A thesis chapter titled "High-performance design features in federated Service-oriented Geographic Information Systems" (available at http://complexity.ucs.indiana.edu/~asayar/thesis/performance4.pdf)