The Ignite Distributed Collaborative Visualization System

Sushil Bhojwani
University of Victoria
sushilsrk@gmail.com

Matt Hemmings
University of Victoria
mhemming@uvic.ca

Dan Ingalls
CDG, SAP
danhhingalls@gmail.com

Jens Lincke
Hasso Plattner Institute
Jens.Lincke@hpi.de

Robert Krahn
CDG, SAP
robertkrahn@gmail.com

David Lary
UT Dallas
dlary@utdallas.edu

Rick McGeer
CDG/US Ignite
rick@mcgeer.com

Glenn Ricart
US Ignite
glenn.ricart@us-ignite.org

Marko Roder
CDG, SAP
markoroeder@cdglabs.org

Yvonne Coady
University of Victoria
ycoady@uvic.ca

Ulrike Stege
University of Victoria
ustege@uvic.ca

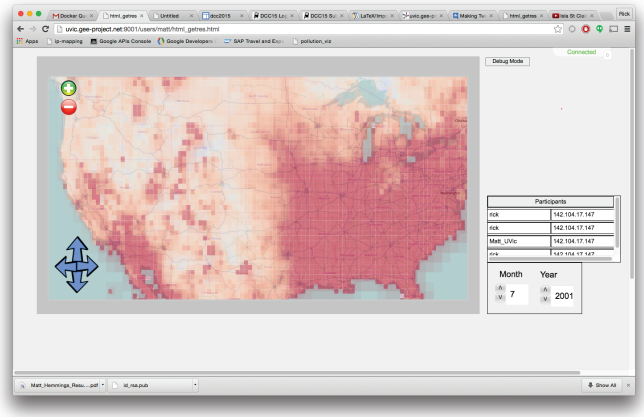

Figure 1: Application Screenshot

Collaboration around large data sets involves interactive visualizations. Both large data sets and complex visualization systems tax the resources of thin clients. Moreover, collaboration requires the transmission of information between widely-separated colleagues. For this reason, distributed visualization systems have focussed on fat client systems interconnected by very high-performance networks. Systems such as the OptIPuter/OptIPortal[3] are very capable, but are not ubiquitous desktop systems. An OptIPortal costs tens of thousands; a gigabit connection to a national research network is thousands/month. In this abstract we show much of this functionality can be brought to the handheld, thanks to the emergence of the Distributed Cloud and the capabilities of HTML5 and the Lively Web.

An interactive application requires a response within 150 ms[6]. Offering an application on a ubiquitous client requires that data and processing take place on a server, and any nontrivial network latency makes it impossible to meet this deadline. The Distributed Cloud becomes an enabler of a new class of application: rich collaboration around large data sets using ubiquitous client devices.

To demonstrate this, we developed an interactive pollution visualizer, with servers distributed across the GENI infrasucture[2], showing pollution values for the entire world on a month-by-month basis over a period of 20 years, on 10-, 25-, 50-, and 100- km^2 grids. The data was calculated from a wide variety of data sources using machine learning[4, 5]. A screenshot of the application appears in Figure 1. Pollution values are displayed in each grid square by a color, with red indicating high intensity. Users of the system are able to manipulate the view using DVR-like controls and map pan and zoom. A unique feature of the application is that multiple users may be looking at this visualization, even on different servers, and jointly manipulating it. The current users are shown in the Participant box in the right side of the screen. When a user manipulates his copy of the map, it changes for all participants. The application becomes an augmented conversation.

With a 150 ms time limit from request to response, the demand on the network is significant. Drawing on the map is a four-step process: the user requests data from the server, the server searches through its database, the server returns the data to the client, and the client draws the points on the map. 30,000 points is a reasonable upper bound on the number of points to be drawn, and experiments determined that modern browsers could draw 30,000 points in 100 ms on a commodity laptop or tablet. This left 50 ms for the other three steps. A tailor-made database with caching could fetch the data in 20 ms.

The only variability is in the network, and this is governed by server/client latency and bandwidth. Our goal was to determine what envelope of network round-trip times and bandwidth would permit us to achieve our goal of a complete referesh within 150 milliseconds. Our wire protocol had about 27 bytes/point, so 30,000 points are roughly 6.4 megabits. We assumed 1500 byte packets and accelerated slowstart (10-packet initial send). We tested four scenarios, as shown below.

The Campus scenario assumed a server on the same campus as the client, with millisecond latency and gigabit bandwidth. City assumed a server within 100 km or so, but not resident on the same campus. Continent assumed an EC-2 like deployment, with a few POPs per continent. World assumes a single POP for the entire globe. Our assumptions were biased towards “Classic Cloud” (Continent and World) deployments. We made a conservative bandwidth assumption for the Campus and generous bandwidth assumptions for the Continent and World scenarios.

The results are shown in Table 1. Only Campus and City scenarios achieved a network time of 30 ms (35 ms for City).

Scenario	Latency	Bandwidth	Request	Response
Campus	1 ms	1 Gb/s	1 ms	8 ms
City	5 ms	1 Gb/s	5 ms	30 ms
Continent	50 ms	100 Mb/s	50ms	30ms
Central	250 ms	100 Mb/s	250ms	1500ms

Table 1: Request and Response Times for Scenarios

We assumed TCP as a transport layer, no loss, and an accelerated slowstart with a 10-packet initial burst. There was no way to achieve design goals without deployments at the City or Campus level.

This analysis sufficed to demonstrate the indispensability of the Distributed Cloud for this application. To recapitulate, simple protocol analysis showed that interactive response time could not be achieved unless a data server was within about 5 ms of the client. A millisecond is roughly 200 km in fiber, so this gives a radius of about 1000 km in straight-line distance. In practice, straight-line distance is a loose lower bound on latency, so a rough guide is about 500 km, or a minimum of 32 POPs across North America for full coverage.

The Ignite Collaborative Visualization System consists of two major components:

1. The Ignite Application Engine, which will be based on the GENI Experiment Engine[1], which offers deployment of applications across the GENI infrastructure, and leverages the commercial ecosystem of lightweight virtualization environments. Specifically, the GEE offers a platform for the deployment of lightweight containers across the GENI infrastructure using Docker.

2. The Lively Web system developed at the Hasso-Plattner Institute, the Communications Design Group at SAP America, and the University of Victoria.

We used the Application Engine to distribute Docker containers with our application across the wide area. In our prototype demo we used GENI Virtual Machines due to temporary restrictions on disk size for Docker containers on the GENI Experiment Engine, which will be our deployment vehicle of choice going forward.

The Lively Web is an integrated environment for unified client- and server-side development of web applications, with an integrated client-server and peer-peer messaging system. It fully abstracts HTML and CSS into a graphical abstraction based on the Morphic system introduced in Self and deployed in Squeak and Scratch. Development of client-side applications is done by a combination of manual drag-and-drop placement and configuration of Morphs, and Javascript programming. Lively integrates Wiki functionality, so deploying the client side of the application is simply a file save on the Wiki server. Lively also comes with a pluggable node.js server programmed from within the browser, so server-side programming is done through the same pane of glass. Lively integrates database servers, and a WebDAV filesystem interface. Lively offers a sophisticated client-server and peer-peer messaging system based on WebSockets, which can be used to invoke Javascript functions remotely: this "Lively2Lively" system forms the basis of our unique remote-control architecture.

The application itself is simply a Lively web page. Data is served from the local server, which employs a quad-tree search across an on-disk database; each leaf of the quadtree is stored as a separate file. The 100-km grid leaf cells are kept permanently in memory to provide rapid access to coarse data. Once a leaf cell at any resolution is read, it is cached in memory; the cache is flushed when the server approaches a pre-set virtual memory limit.

Messaging is based on Lively2Lively. The application page declares a local map drawing service, and registers itself as a conversation participant to a centralized server. On each map update, it broadcasts an update message to other participants, with Lively2Lively handling inter-server message delivery. On receipt of an update message, the page requests data from the local server corresponding to the map viewbox contained in the message, recenters the map to that viewbox, and then draws the points on receipt from the server. The workflow (and the perceived latency) is thus identical from a remote update and a UI event. A user can't distinguish latency from a remote participant without out-of-band communication. Though there is speed-of-light latency between participants, the time between the notification to a user that a remote participant has changed the map and its change is within the 150 ms bound.

The system was a major International Demonstrator in the plenary session at the SmartFuture2015 event, with participants in Washington, Tokyo, San Francisco, Victoria, Canada, Potsdam, Germany, and Ghent, Belgium. Participants used a large number of devices in the demonstration, including an Android phone and iPhone, a Chromebook, an iPad, and several laptop systems. All devices performed to the 150 ms bound and all participants saw the same display.

This work was partially supported by the GENI Project Office under contract 1957, by SAP America and by MITACS. The authors wish to thank the staff at the GENI project office and the participants in the demonstrator.

1. REFERENCES

[1] A. Bavier, J. Chen, J. Mambretti, R. McGeer, S. McGeer, J. Nelson, P. O'Connell, G. Ricart, S. Tredger, and Y. Coady. The geni experiment engine. In *Teletraffic Congress (ITC), 2014 26th International*, pages 1–6, Sept 2014.

[2] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar. Geni: A federated testbed for innovative network experiments. *Computer Networks*, 61(0):5 – 23, 2014. Special issue on Future Internet Testbeds - Part I.

[3] T. A. DeFanti et al. The optiportal, a scalable visualization, storage, and computing interface device for the optiputer. *Future Generation Computer Systems*, 25(2):114 – 123, 2009.

[4] D. J. Lary, F. S. Faruque, N. Malakar, A. Moore, B. Roscoe, Z. L. Adams, and Y. Eggelston. Estimating the global abundance of ground level presence of particulate matter (pm2. 5). *Geospatial health*, 8(3):611–630, 2014.

[5] D. J. Lary, T. Lary, and B. Sattler. Using machine learning to estimate global pm2.5 for environmental health studies. *Environmental Health Insights*, 1(doi: 10.4137/EHI.S15664):41–52, 2015.

[6] N. Tolia, D. G. Andersen, and M. Satyanarayanan. Quantifying interactive user experience on thin clients. *Computer*, 39(3):46–52, 2006.