ON-DEMAND HIGH-PERFORMANCE VISUALIZATION OF SPATIAL DATA ON HIGH-RESOLUTION TILED DISPLAY WALLS

Tor-Magne Stien Hagen, Daniel Stødle and Otto J. Anshus
Department of Computer Science, Faculty of Science and Technology, University of Tromsø, Norway

Keywords: Visualization, High-resolution tiled display walls, Live data sets, On-demand computation.

Abstract: Visualization of large data sets on high-resolution display walls is useful and can lead to new discoveries that would not have been noticeable on regular displays. However, exploring such data sets with interactive performance is challenging. This paper presents live data sets, a scalable architecture for visualization of large data sets on display walls. The architecture separates visualization systems from compute systems using a live data set containing data customized for the particular visualization domain. Experiments conducted show that the main bottleneck is the compute resources producing data for the visualization side. When all data is cached in the live data set, the main bottleneck (decoding images to create OpenGL textures and constructing geometry from raster data) is on the visualization side. On a 22 megapixel, 28 node display wall, the visualization system can decode 414.2 megapixels of images (19 frames) per second. However, the decoding is multi-threaded, and increased performance is expected using multi-core computers.

1 INTRODUCTION

This paper presents live data sets, a new architecture for interactive visualization of images, maps, the Earth and other data sets on tiled display walls. A display wall is a high-resolution wall-sized display typically built by tiling many projectors. Each projector is driven by a computer called a display node (Fig. 1). Traditional approaches to computation and visualization of spatial data on display walls use a three-tier display-compute-data architecture (Zhang, 2008; Jeong et al., 2006; Smarr et al., 2003; Correa et al., 2007) where display nodes communicate directly with the compute resources. Live data sets use a data space architecture where display nodes communicate with compute resources through a data set. This yields several advantages. First, the display and compute sides become simpler since the logic for communication between compute and rendering nodes is hidden by the data set. Second, the data set can be close to the rendering side, which reduces latency and makes higher bandwidth possible. Finally, the data set remains accessible from the display side also in cases where the compute resources are offline.

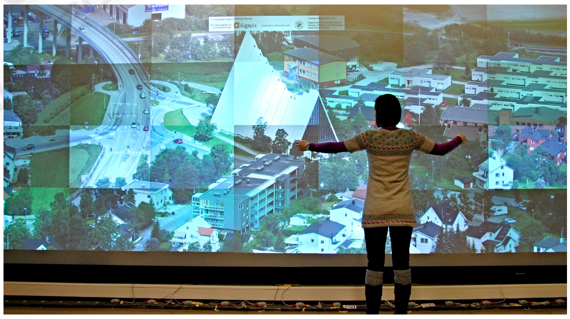

Figure 1: A user navigates in a 13.3 gigapixel image on a 22 megapixel tiled display wall using hand- and arm-gestures.

In the live data set architecture the data set requests data from compute resources on-demand. This is in contrast to existing systems such as Google Earth where the data set is pre-processed by the server side. On-demand computation and revalidation of data ensures that the display side always has the latest version of the original data sets used. Further, on-demand computation and acquisition of data reduces storage requirements since data is collected and computed only when needed. The live data set architecture requires no custom code running on the compute side to communicate with the display resources. The data set adapts to the protocol used by compute resources and

Stien Hagen T., Stødle D. and J. Anshus O. (2010).
ON-DEMAND HIGH-PERFORMANCE VISUALIZATION OF SPATIAL DATA ON HIGH-RESOLUTION TILED DISPLAY WALLS.
In *Proceedings of the International Conference on Imaging Theory and Applications and International Conference on Information Visualization Theory and Applications*, pages 112-119
DOI: 10.5220/0002849601120119

not the other way around. Additionally, the system can utilize remote compute and data resources like GRIDs and supercomputers where outgoing connections are often administratively prohibited.

WallScope is an implementation of the live data set architecture. WallScope comprises visualization systems, live data sets and compute/data resources. The visualization systems run a client on each computer in the display wall's display cluster. Each client requests data from the live data set which it then uses as part of the rendering. The data set initiates a local or remote computation to satisfy the client's request, or returns a cached copy of the request if the computation has been performed recently. To manage the visualization systems' view, a central state server is used which broadcasts a heartbeat message at a configurable rate to update each client's view of the visualization's state.

This paper makes three main contributions: (i) live data sets, a new architecture for visualization of high-resolution data sets on tiled display walls; (ii) WallScope, a system realizing the live data set architecture; and (iii) an evaluation of the system documenting its performance characteristics.

2 RELATED WORK

Spatial data, and in particular geographic information, can be accessed and visualized using numerous software applications. Table 1 summarizes the differences between WallScope and related work.

Google Earth (Chang et al., 2006) is a popular geographic information system. It is implemented using the BigTable system on the server side. BigTable is a distributed storage system for managing structured data. It uses the Google File System (GFS) (Ghemawat et al., 2003) to store its data. For Google Earth, BigTable uses one table to pre-process data, and a different set of tables for serving client data. The pre-processing pipeline uses MapReduce (Dean and Ghemawat, 2008) to transform data. Google Earth has been shown running on a display wall (Williams, 2007). The system uses Chromium (Humphreys et al., 2002) to distribute rendering primitives from one central computer running Google Earth, to software running on each display cluster node. Chromium has scalability issues as the network often becomes a bottleneck (Kunz et al., 2003). Google Earth can also run on the HIPerWall (Calit2, 2008), using one or more instances per display cluster node. The instances are controlled using Google Earth's API. A controller receives input from a user and forwards the resulting view state to the instances. This solution does not support on-demand computation of domain specific data, and the system's performance is not documented.

ArcGIS Engine (ESRI, 1997) is a collection of components that are used to create custom GIS applications. One such system (Liang et al., 2007) uses ArcGIS Engine to do parallel map rendering on a tiled display wall. This system uses a master node and six rendering nodes, all running ArcGIS Engine and connected to a back-end GIS database. The master shows the full scene of the data set and takes control input from a user. The rendering nodes retrieve layer data from the back-end and view information from the master. However, in contrast to WallScope this system has no separate compute resources. Additionally, the ArcGIS components are only available for Windows.

Tellurion (Kooima, 2008) is a planetary visualizer with real-time capabilities for data-manipulation. Using a GPU centric approach the system is able to combine disparate planetary-scale data sets to produce a uniform composite visualization of them. The visualizer has been ported to several display walls. Although the visualizer has real-time capabilities for blending data sets with different projections, the data sets have to be downloaded to disk in advance. WallScope uses on-demand fetching and revalidation of data to ensure that the system is using the latest version of real-time data sets.

Several other visualization systems can transform and visualize spatial data, including Active Data Repository (Kurc et al., 2001), DataCutter (Beynon et al., 2001), Active Semantic Caching (Andrade et al., 2007), Scalable Parallel Visual Networking (Correa et al., 2007), ParVox (Li, 2002), OptiStore (Zhang, 2008), OptiPuter (Taesombut et al., 2006), The Digital Light Table (Katz et al., 2004), Multi-Surface Light Table, The Remote Interactive Visualization and Analysis System (Li et al., 1996) and The Scalable Adaptive Graphics Environment (Jeong et al., 2006), based on TeraVision (Singh et al., 2004) and TeraScope (Zhang et al., 2003). However, these systems have tightly integrated the compute resources with the display resources, and require both the display side and the compute side to run custom code. This limits the compute capabilities to local compute resources. Additionally most systems require the data to be located on the compute nodes' disks. WallScope extends this by utilizing a live data set to provide the rendering applications with domain specific data from local and remote data and compute resources, transparent to the display side of the system.

Table 1: WallScope compared to visualization and compute systems presented in the litterature.

Visualization system	**Separate Compute Resources**	**Remote Compute Resources**	**On-Demand Computation**
WallScope	Yes	Yes	Yes
Google Earth / Maps (BigTable)	Yes	No	No
ArcGIS (Display Wall Version)	No	No	No
Active Data Repository	Yes	No	Yes
DataCutter	Yes	No	Yes
Scalable Parallel Visual Networking	Yes	No	Yes
ParVox	Yes	No	Yes
Tellurion	No	No	No
OptiStore	Yes	No	Yes
OptiPuter	Yes	No	Yes
Digital Light Table	Yes	No	Yes
Remote Interactive Visualization and ...	Yes	No	Yes
Multi-Surface Light Table	Yes	No	Yes
Scalable Adaptive Graphics Environment	Yes	No	Yes

3 ARCHITECTURE

Fig. 2 shows the WallScope architecture. A visualization client runs on each display cluster node. The view state of the clients is provided by a state server. Each client requests data from the live data set which it then uses for the rendering. The live data set returns the data immediately if it is available. Otherwise, a processing message is sent to local or remote compute resources that generate domain specific data for the visualization system. The compute resources generate customized data according to the client request. For local compute resources, the request is satisfied using two caches, the first containing original data and the second containing pre-processed data. The WallScope architecture enables load-balancing by separating compute intensive tasks from the visualization clients and executing them on the backend compute resources. This makes it possible to add compute resources as needed, and not limit the system's compute power to the number of display cluster nodes. Data with realtime properties may pass through the system without modification, reducing latency for this kind of data.

4 DESIGN

The WallScope visualization systems execute as a set of clients, one client per display node. Each client is responsible for requesting and displaying its part of the output image calculated from its position in the display wall grid. A state server maintains the state and updates all clients at a configurable rate using a heartbeat state message. The clients combine the state received in the heartbeat message with their local view frustum to calculate the data they need for rendering. The state server accepts user input which is merged into the state and propagated to the clients.

The live data set stores all data processed by WallScope. WallScope uses a single centralized live data set to hide data fetching and computing to the visualization systems. However, the live data set can become a bottleneck if the load from the clients exceeds the processing speed of the computer running the live data set, or if the network between the data set and the clients is saturated. For this reason each display node also comprises a local cache. The local caches support different configurations: (i) local caching; (ii) peer-to-peer exchange of cached data; and (iii) no caching at all. The latter approach is preferred if neither the network nor the computer running the live data set is a bottleneck, because it limits the amount of duplicate entries caused by having objects cached in both the local cache clients and the visualization system's clients.

The local compute resources execute as a set of clients on one or several compute clusters. Each compute node comprises an on-demand cache of original data and a cache of pre-processed data. The compute nodes receive processing messages from the live data set. From the processing message the compute engine requests original and pre-computed data from its local caches and generates data according to the request. The local compute system uses a single on-demand centralized cache of original data between the Internet and the local compute resources. This de-

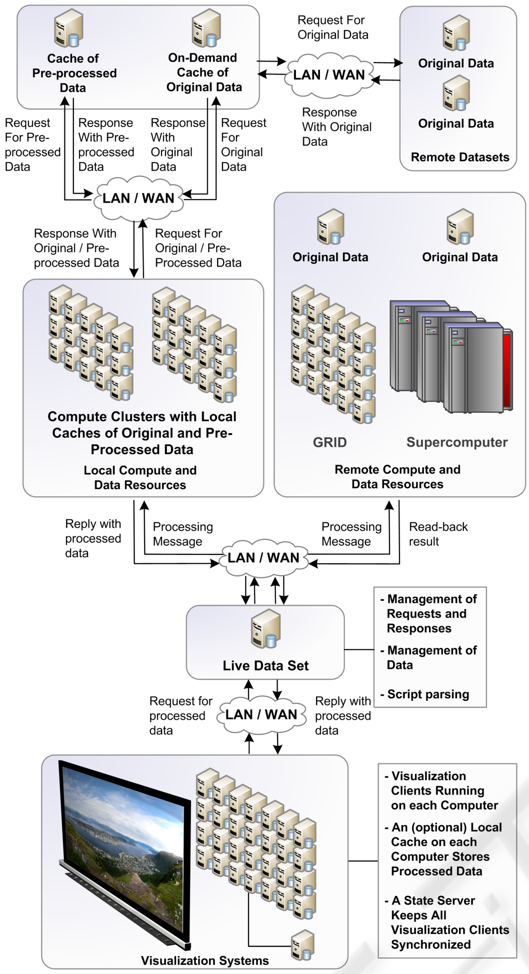

Figure 2: The WallScope architecture. The live data set provides visualization clients with customized data from local and remote data and compute resources.

sign choice was made to: (i) make the system appear as a single entity to the outside world; (ii) limit the number of outgoing connections to prevent external servers from being overloaded or banning WallScope due to excessive resource usage; and (iii) authenticate and keep track of external session state required by external servers. The cache of pre-processed data contains pre-downloaded and pre-computed data that is pre-processed for the local compute resources. For example to reduce latency, reduce number of passes over the data, or for converting data to a more efficient memory layout.

5 IMPLEMENTATION

Two visualization systems have been implemented for the WallScope system. WallGlobe is a visualization system for planetary scale data sets. It is implemented in Java, using Java OpenGL (JOGL) for rendering. WallView is a visualization system for gigapixel images, similar to Seadragon[1]. It is implemented in C++ and uses OpenGL for rendering. Both visualization systems use a sort-first rendering approach and run on Linux, Windows and Mac OS X. Users can pre-fetch data to the live data set by running these visualization systems on their laptops.

The main components of the visualization clients are the rendering engine, the request queue and the request threads. Although OpenGL is a thread-safe API, most implementations are not. Therefore the rendering engines have a single thread of execution.

The rendering thread draws at a configurable number of iterations per second. The thread performs view frustum clipping and back-face culling before it requests data for all the objects that are visible using the request queue. Request threads fetch data from this queue and notifies the rendering thread as soon as the data is downloaded and decoded.

Every computer driving the display wall executes a local copy of the visualization clients. A state server accepts input from external controllers and broadcasts this state to the other clients using UDP multicast.

The live data set and the Internet cache of original data are implemented using Squid. Squid is a high-performance web caching proxy (Wessels et al., 1995). Squid was chosen because it is a open source, highly configurable, cross-platform and several Squid servers can be configured to co-operate in various cache hierarchies.

The local compute resources execute a custom map system implemented in Common Lisp. Each node uses two different caches. A web cache of original data realized using Squid, and a cache of pre-processed data realized using the network file system (NFS) and memory mapping. The pre-computed cache is indexed and memory mapped when the compute application is initially started. Files in this cache are not updated when WallScope executes, as opposed to the on-demand cache of original data which might invalidate and download new content during execution.

6 EXPERIMENTS

To measure the performance of WallScope three experiments were conducted. The first experiment measured the time used to request 900 512x512 image tiles from one display node, with the purpose of iden-

[1] http://www.seadragon.com/

tifying performance characteristics when data is accessed from different locations in the architecture. The second experiment measured the speedup when going from 1 to 26 compute nodes, to understand the scalability of the live data set and the local compute resources. The third experiment used a WallGlobe camera trace to isolate bottlenecks in the WallScope architecture, and measure the system's performance under different configurations and loads.

6.1 Methodology

The hardware used in the experiments was: (i) a 28 node display cluster (Intel P4 EM64T 3.2 GHz, 2GB RAM, Hyper-Threading, NVidia Quadro FX 3400 w/256 MB VRAM) interconnected using switched Gigabit Ethernet and running the 32-bit version of the Rocks Linux cluster distribution 4.0; (ii) a computer running the state server; (iii) a computer running the live data set; and (iv) a 26 node compute cluster. Each display node was connected to a projector with a resolution of 1024x768 pixels, arranged in a 7x4 grid for a total of 7168x3072 pixels (22 megapixels). The state and live data set computers had the same specifications as the display cluster nodes. The compute nodes also had the same specifications, but were running the 64-bit version of Rocks.

To conduct the first experiment, a custom Java application was written to perform 900 image requests. The following statistics were measured: (i) the time used to compute images on one compute node; (ii) the time used to read the processed data from the live data set to one display node; and (iii) the time used to read the processed data from the display node's local cache with and without the time used to decode the images at the display node. The 900 image tiles requested during the experiment correspond to 236 megapixels, in total 31.84 megabytes of JPEG image data. Each image tile used the Landsat data set as the base layer with the ocean masked with data from the Blue Marble data set. The original data sets that comprised the processed images were pre-fetched to the original data cache. The vector data used in the masking of the ocean was replicated from the centralized cache of pre-processed data to every compute node.

For the second experiment, the speedup was measured when going from 1 to 26 compute nodes. The experiment used the same 900 image requests. All requests were divided between the computers round-robin.

The third experiment used a camera trace played back at varying speeds. The trace consisted of a set of waypoints. Each waypoint described the position and rotation of the camera at the waypoint. The camera's position and rotation was then interpolated over a spline curve calculated between the waypoints. The trace started with the entire Earth visible in the view frustum, and then zoomed into a specific part of the Earth. All waypoints were picked in such a way that the camera only visited new tiles. The time spent interpolating between each waypoint was 30, 25, 20, 15, and 10 to 1 seconds. The WallGlobe state server was configured to multicast the global state derived from the camera trace to each of the WallGlobe clients 50 times per second (matching the refresh rate of the projectors). The total number of requests generated by the trace was 8585. 5111 (59.53%) of these requests were requests for images and 3474 (40.47%) were requests for elevation data. All the image tiles were 512x512 pixels. This corresponds to approximately 1340 megapixels of image updates. All waypoint times were repeated over different WallScope cache- and compute-configurations. The configurations were: (i) full local caches; (ii) a live data set containing all requested data; and (iii) computing on 1, 2, 4, 8, 16 and 26 nodes.

6.2 Results

Table 2 shows the measured times for experiment 1. The first column shows the location of the requested data. The second column shows the time used to complete all the 900 requests. The third column shows the mean time per request.

Table 2: Time used to request 900 512x512 pixel (236 megapixels) image tiles.

Data Location	Time Used	Mean
Compute Node	296.238 sec	329.2 ms
Live data set	1.411 sec	1.6 ms
Local Cache	0.716 sec	0.8 ms
Local Cache (w/dec.)	13.850 sec	15.4 ms

Fig. 3 shows the result of experiment 2. In the figure the actual speedup is plotted against a linear speedup.

Fig. 4, 5, 6 and 7 show the result of experiment 3. Fig. 4 shows the displayed requests from each of the experiment configurations. The y-axis is the number of requests that were displayed during the experiment. A "displayed request" is a request that was loaded in memory and displayed at least one frame during the experiment. The x-axis is the seconds used between waypoints. The upper bold line is the total number of requests generated by the trace.

Fig. 5 shows the number of completed requests during the experiment. This is the sum of the displayed requests and the requests that were down-

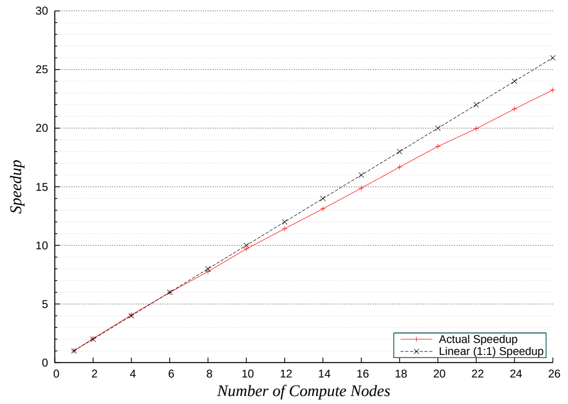

Figure 3: Speedup when going from 1 to 26 compute nodes.

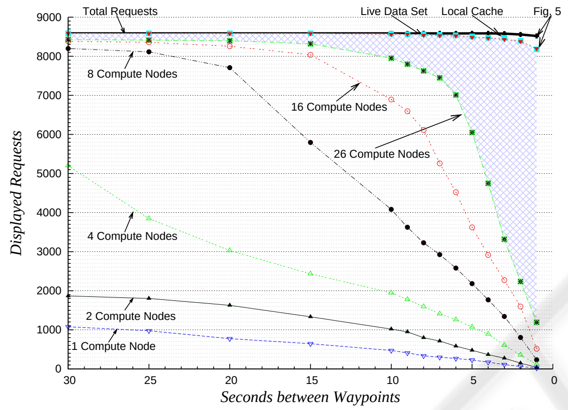

Figure 4: The number requests that were loaded into memory and contributed to at least one frame.

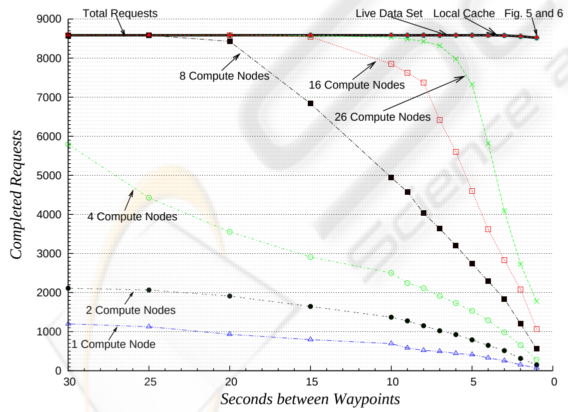

Figure 5: The total number of completed requests. (The displayed and non-displayed requests).

loaded and decoded but not displayed because the planet's source tile of the request was outside the view-frustum at the time the data was available.

Fig. 6 shows the requested, completed and displayed requests for the full local cache configuration using one second between waypoints (marked by an

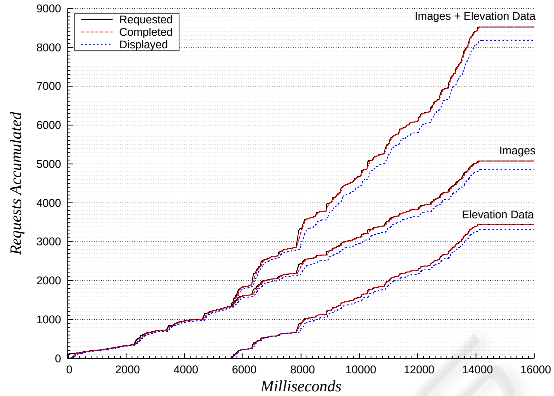

Figure 6: The cumulative number of requested, completed and displayed requests with full local caches. The time between each waypoint is 1 second.

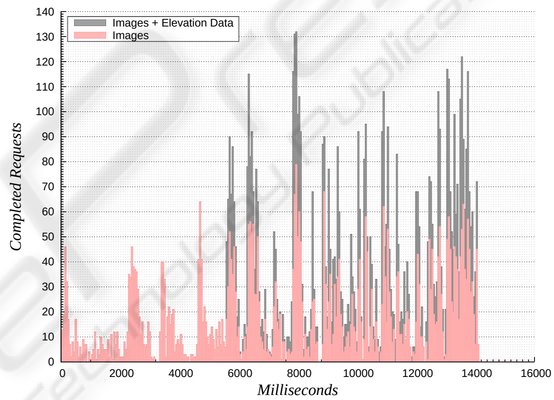

Figure 7: The number of completed requests for the full local cache configuration using one second between each waypoint. Each of the requests are stacked in 50 milliseconds intervals.

arrow in the upper right corner of both Fig. 4 and Fig. 5). The y-axis is the number of requests accumulated and the x-axis is the time in milliseconds.

Fig. 7 shows the completed requests of the same trace stacked in intervals of 50 milliseconds, also marked in Fig. 5.

6.3 Discussion

The time to request data from the local cache is twice as fast as requesting data from the live data set. However, the time used to request and decode images on one node is 13.85 seconds. This is over 9 times slower than just requesting the images from the live data set, and therefore contributes to a much larger part of the overall time, compared to local versus central storage. Requesting data stored in the live data set is two orders of magnitude faster than computing the data on one compute node. For each processed image tile the compute node must request the original image tiles that comprise the processed image tile. This requires data from at least one image tile from the Landsat data set and one image tile from the Blue Marble data set. Further, the compute node must create a raster surface matching the specification from the client request (512x512 pixels) and fill the raster surface with data from both data sets based on the meta information from the vector shape data. During decoding of JPEG images the display node had a CPU load of 100%.

From 1 to 6 compute nodes the system has a near linear speedup, as shown in Fig. 3. However, when the number of nodes increases beyond 6, the speedup is slightly reduced. This is caused by the round-robin work distribution. Because each data request has varying amounts of ocean to mask, the workload for each work request varies. Work is handed out in a round-robin fashion without any feedback mechanism. Therefore some of the nodes will get more work than others, which is why the performance does not increase linearly with the number of compute nodes added.

During experiment 3 all WallGlobe clients kept the same framerate as the refresh rate between the computer and the display. This shows that the sort-first approach used by WallGlobe is sufficient for driving the 22 megapixel display wall. Fig. 4 shows that the main bottleneck of the system is the computation of customized data, illustrated by the difference between the graphs when everything is computed and the graphs when everything is stored in the live data set or the local cache. As the graphs illustrate the system benefits from increased number of compute nodes. Documenting performance characteristics of different cache replacement policies is outside the scope of this paper. However, the shaded area on the graph is where the expected performance of the system would be using all compute nodes with caching enabled. Different cache replacement policies will result in a system performance that will fall within this area.

Fig. 4 and Fig. 5 show that the difference between local caching on each node and central storage using the live data set on one node is small. Thus neither the network nor the live data set is a bottleneck of the system. From 8 to 1 seconds the system does not display all requests, despite the fact that they are stored in the local cache (Fig. 4). This is explained by the time used to decode JPEG images to create OpenGL textures, and the time used to parse elevation data to create geometry in the WallGlobe clients (Fig. 6). At 8000 milliseconds the trace generates more requests than the WallGlobe clients are able to display, illustrated by the graphs showing displayed requests for both image and elevation data. When the request threads can't keep up with the frequency of the updates, the amount of displayed data decreases. At 8000 milliseconds the completed requests peeks at 132 requests (Fig. 7). This corresponds to 2640 requests per second (132 x (1000 ms / 50 ms)). Of these 132 requests 79 are requests for images corresponding to a peak rate of 1580 images per seconds (414.2 megapixels/s) the equivalent of 18.83 updates per second on the 22 megapixel display wall. As decoding of JPEG images and parsing of elevation data is performed in separate request threads, the system will benefit from multi-core CPUs.

7 CONCLUSIONS

This paper presented live data sets, a new architecture for interactive visualization of images, maps, the Earth and other data sets on high resolution tiled display walls. To demonstrate live data sets WallScope was built. By separating data processing from display rendering and using local and remote data- and compute-resources to provide on-demand customized data for the particular visualization domain, WallScope's visualization capabilities are not limited to the processing power of the display resources. The experiments conducted show that the sort-first rendering approach used by the visualization systems combined with a simple state server enables each visualization client to keep the same framerate as the refresh rate between the computer and the display. When visualizing the Earth by combining data from the Landsat data set with data from the Blue Marble data set, the bottleneck of the system is the merging process of the data sets on the compute nodes. However, the time used to combine data sets decreases by a factor of 23 when increasing the number of compute nodes from 1 to 26. When all data is stored, the bottleneck of the system is decoding JPEG images to create OpenGL textures and processing geometry from elevation data. The visualization system can produce 414.2 megapixels per second, resulting in 19 decoded frames per second. However, the decoding is multi-threaded, and higher framerates are expected using multi-core computers. This tracks the current trend towards more CPU cores.

ACKNOWLEDGEMENTS

The authors wish to thank Espen Skjelnes Johnsen, Lars Ailo Bongo and Joseph Ricci for discussions, as well as the technical staff at the CS department at the

University of Tromsø. The authors also wish to thank Eirik Helland Urke for providing the 13.3 gigapixel image. This work has been supported by the Norwegian Research Council, projects No. 159936/V30, SHARE - A Distributed Shared Virtual Desktop for Simple, Scalable and Robust Resource Sharing across Computer, Storage and Display Devices, and No. 155550/420 - Display Wall with Compute Cluster.

REFERENCES

Andrade, H., Kurc, T., Sussman, A., and Saltz, J. (2007). Active semantic caching to optimize multidimensional data analysis in parallel and distributed environments. *Parallel Comput.*, 33(7-8):497–520.

Beynon, M. D., Kurc, T., Çatalyürek, U., Chang, C., Sussman, A., and Saltz, J. (2001). Distributed processing of very large datasets with datacutter. *Clusters and computational grids for scientific computing*, 27(11):1457–1478.

Calit2 (2008). http://hiperwall.calit2.uci.edu/?q=node/1.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Ch, T., Fikes, A., and Gruber, R. E. (2006). Bigtable: A distributed storage system for structured data. In *In Proceedings of the 7th Conference on USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, pages 205–218.

Correa, W. T., Klosowski, J. T., Morris, C. J., and Jackmann, T. M. (2007). SPVN: a new application framework for interactive visualization of large datasets. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, page 6.

Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113.

ESRI (1997). http://www.esri.com/software/arcgis/.

Ghemawat, S., Gobioff, H., and Leung, S. T. (2003). The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43.

Humphreys, G., Houston, M., Ng, R., Frank, R., Ahern, S., Kirchner, P. D., and Klosowski, J. T. (2002). Chromium: a stream-processing framework for interactive rendering on clusters. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 693–702.

Jeong, B., Renambot, L., Jagodic, R., Singh, R., Aguilera, J., Johnson, A., and Leigh, J. (2006). High-performance dynamic graphics streaming for scalable adaptive graphics environment. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 108.

Katz, D., Bergou, A., Berriman, G., Block, G., Collier, J., Curkendall, D., Good, J., Husman, L., Jacob, J., Laity, A., Li, P., Miller, C., Prince, T., Siegel, H., and Williams, R. (2004). Accessing and visualizing scientific spatiotemporal data. In *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*, pages 107–110.

Kooima, R. (2008). *Planetary-scale Terrain Composition*. PhD thesis, Computer Science, Graduate College of the University of Illinois, Chicago.

Kunz, A., (editors), J. D., Staadt, O., Walker, J., Nuber, C., and Hamann, B. (2003). A survey and performance analysis of software platforms for interactive cluster-based multi-screen rendering.

Kurc, T., Çatalyürek, U., Chang, C., Sussman, A., and Saltz, J. (2001). Visualization of large data sets with the active data repository. *IEEE Comput. Graph. Appl.*, 21(4):24–33.

Li, P. (2002). Supercomputing visualization for earth science datasets. In *Proceedings of 2002 NASA Earth Science Technology Conference*.

Li, P., Duquette, W. H., and Curkendall, D. W. (1996). Riva: A versatile parallel rendering system for interactive scientific visualization. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):186–201.

Liang, H., Arangarasan, R., and Theller, L. (2007). Dynamic visualization of high resolution gis dataset on multi-panel display using arcgis engine. *Computers and Electronics in Agriculture*, 58(2):174 – 188.

Singh, R., Jeong, B., Renambot, L., Johnson, A., and Leigh, J. (2004). Teravision: a distributed, scalable, high resolution graphics streaming system. In *CLUSTER '04: Proceedings of the 2004 IEEE International Conference on Cluster Computing*, pages 391–400.

Smarr, L. L., Chien, A. A., DeFanti, T., Leigh, J., and Papadopoulos, P. M. (2003). The optiputer. *Commun. ACM*, 46(11):58–67.

Taesombut, N., Wu, X. R., Chien, A. A., Nayak, A., Smith, B., Kilb, D., Im, T., Samilo, D., Kent, G., and Orcutt, J. (2006). Collaborative data visualization for earth sciences with the optiputer, future generation computer systems. *Future Gener. Comput. Syst*, 22:955–963.

Wessels, D., Claffy, K., and Braun, H.-W. (1995). NLANR prototype Web caching system, http://ircache.nlaur.net/.

Williams, C. (2007). http://blog.irisink.com/2007/06/14/display-wall-with-google-earth-on-mac-os-x/.

Zhang, C. (2008). *OptiStore: An On-Demand Data processing Middleware for Very Large Scale Interactive Visualization*. PhD thesis, Computer Science, Graduate College of the University of Illinois, Chicago.

Zhang, C., Leigh, J., DeFanti, T. A., Mazzucco, M., and Grossman, R. (2003). Terascope: distributed visual data mining of terascale data sets over photonic networks. *Future Gener. Comput. Syst.*, 19(6):935–943.