A Step towards Making Local and Remote Desktop Applications Interoperable with High-Resolution Tiled Display Walls

Tor-Magne Stien Hagen, Daniel Stødle, John Markus Bjørndalen, Otto Anshus

A Step Towards Making Local and Remote Desktop Applications Interoperable with High-Resolution Tiled Display Walls

Tor-Magne Stien Hagen, Daniel Stødle, John Markus Bjørndalen, and Otto Anshus

Department of Computer Science, Faculty of Science and Technology, University of Tromsø
{tormsh,daniels,jmb,otto}@cs.uit.no

Abstract. The visual output from a personal desktop application is limited to the resolution of the local desktop and display. This prevents the desktop application from utilizing the resolution provided by high-resolution tiled display walls. Additionally, most desktop applications are not designed for the distributed and parallel architecture of display walls, limiting the availability of such applications in these kinds of environments. This paper proposes the Network Accessible Compute (NAC) model, transforming personal computers into compute services for a set of display-side visualization clients. The clients request output from the compute services, which in turn start the relevant personal desktop applications and use them to produce output that can be transferred into display-side compatible formats by the NAC service. NAC services are available to the visualization clients through a live data set, which receives requests from visualization nodes, translates these to compute messages and forwards them to available compute services. Compute services return output to visualization nodes for rendering. Experiments conducted on a 28-node, 22-megapixel, display wall show that the time used to rasterize a 350-page PDF document into 550 megapixels of image tiles and display these image tiles on the display wall is 74.7 seconds (PNG) and 20.7 seconds (JPG) using a single computer with a quad-core CPU as a NAC service. When increasing this into 28 quad-core CPU computers, this time is reduced to 4.2 seconds (PNG) and 2.4 seconds (JPG). This shows that the application output from personal desktop computers can be made interoperable with high-resolution tiled display walls, with good performance and independent of the resolution of the local desktop and display.

1 Introduction

A display wall is a wall-sized high-resolution tiled display. It provides orders of magnitude higher resolution than regular desktop displays and can provide insight into problems not possible to visualize on such displays. The large size of display walls enable several users to work on the same display surface, either to compare visualizations or to collaborate on the same visualization. The combination of resolution and size enable users to get overviews of the visualizations, at the same time being able to walk up close to look at details.

Visualization domains that benefit from the resolution offered by display walls include gigapixel images and planetary-scale data sets. These types of domains provide

content in the order of tens and thousands of megapixels. In addition, more ”standard” visualization domains such as spreadsheet, word-processing, and presentation-style applications can benefit from higher resolution displays, enabling them to display much more content than a normal sized display would allow for.

However, applications are tied to both the resolution of the local desktop and display, and the operating system environment installed on the local computer. In addition, display walls often comprise a parallel and distributed architecture, which often requires parallelizing applications to run them with good performance. This makes porting open-source software time-consuming and proprietary software solutions close to impossible. For example, showing a Microsoft Word document on a display wall is difficult, since Word is designed for a single computer system, and therefore cannot be simply “run” on the display wall. A modified remote desktop system can be used to bring the content of a computer display to a display wall. However, the resolution of the remotely displayed content usually matches the resolution of the local computer's display. Although, some remote desktop systems support higher virtual resolution (such as the Windows Remote Desktop Protocol (RDP) [17] supporting a maximum resolution of 4096x2048), they still do not utilize the full resolution of typical display walls, and such systems have performance problems with increasing number of pixels [15]. In addition, some desktop applications have a predefined layout for the graphical user-interface. For example, it is to the authors' knowledge not any PDF viewer that can show more than a couple of PDF pages in width. For regular resolution displays this might be enough, but for high-resolution tiled display walls, which often have orders of magnitude higher resolution, it is not.

To address these problems, this paper presents the Network Accessible Compute (NAC) model, transforming compute resources into compute services for a set of visualization clients (figure 1). The NAC model defines two classes of compute resources; static such as clusters, grids and supercomputers and dynamic such as laptops and desktop computers. Static compute resources are accessed according to their security policies and access protocols. Dynamic compute resources are customized, on-the-fly, to become compute services in the system. The dynamic compute resources are the main focus of this paper. A live data set [8] separates the compute-side from the display-side, thus enabling both compute services and visualization clients to be added or removed from the system without affecting their underlying implementation or communication protocols. This situation is different from a traditional client-server model. Firstly, compute services are communicating with the visualization nodes through a data space architecture allowing both visualization- and compute-nodes to be added transparently to each other. Secondly, for dynamic compute resources, users have their own software environment installed on the compute service, which enables the compute-side to produce customized data for the display-side based on users custom software installation. Visualization systems can therefore visualize this data without understanding the original data format, as long as a transformation function exists that can represent the data in a format familiar and customized to the visualization system.

Using personal desktop computers as compute services for a display wall tracks the current trend in computer hardware architectures. Today, modern computers have become both multi- and many-core. The increase in transistor density combined with

Fig. 1. Illustration of multiple desktop computers used as a NAC resource to provide processed data for a visualization system running on a high-resolution tiled display wall.

the memory-, ILP- and frequency-walls [16] has forced processor vendors into devoting transistors to CPU cores, on-chip caches and memory- and communication-systems, rather than extracting instruction level parallelism or increasing frequency on single cores [24]. Current contemporary processor chips contain multiple cores up to 100 per chip [27]. Current state of the art GPUs contain up to 480 cores per chip [19]. Following Moores law, there is no indication that this trend will not continue for the foreseeable future. Users own more and more computers, and some might have available processing, memory, storage and network available.

The NAC model improves on the following: (i) It enables desktop computers to produce data for a display wall without modifying or porting the applications on the desktop computer; (ii) it enables remote compute resources to produce data for a display wall without requiring custom software running on the remote site; (iii) it allows for visualization of data from desktop computers without being limited to the resolution of the local display; (iv) it enables cross-platform visualization of data located on a desktop computer without the need for the application to be executed on the visualization node; and (v) desktop computers are customized by a live data set and do therefore not need to install or keep any software updated to be able to communicate with the display-side.

The novelty of the system is the usage of locally installed desktop applications in a display wall context, by decoupling the resolution of the local computer from the display, thus enabling existing desktop applications to utilize the resolution of high-resolution tiled display walls.

This paper makes three contributions: (i) The Network Accessible Compute (NAC) model; (ii) WallScope, a system realizing the NAC model in a personal computing environment; and (iii) a performance evaluation of the WallScope system.

2 Related Work

The NAC model has common characteristics with public (global) computing, where the idea is to use the world's computational power and disk space to create virtual supercomputers capable of solving problems and conduct research previously infeasible. There are a number of projects focusing on public computing, among others SETI@home [3], Predictor@home [21], Folding@home [20] and Climateprediction.net [25]. These projects use the BOINC (Berkeley Open Infrastructure for Network Computing) [2] platform. The overall goal of BOINC is to make it easy for scientists to create and operate public-resource computing projects. A user wanting to participate in the BOINC project downloads and installs a BOINC client which is used to communicate with the server-side. While there are similarities between the NAC model and BOINC there are some important differences. In BOINC the focus is to make it easy to utilize available computational resources. For the NAC model, the focus is to utilize desktop applications for domain specific computation of data for a set of visualization clients. NAC gives users complete control over what data is shared, and enables users to choose this data from their personal computer on a per data-element basis, for example only page 1 and 3 of a 10-page document. This also includes complete control over the output format such as pixels, PDF, original source, etc. In addition, the live data set used as part of the realization of the NAC model supports local and remote compute resources like clusters and supercomputers, which are not supported by BOINC focusing exclusively on public computing.

There are other system sharing characteristics with NAC such as Condor [14], Minimum Intrusion Grid (MIG) [28], XtremWeb [6] and XtremWeb-CH [1] (comprising the two versions XWCH-sMs, and XWCH-p2p). However, their main difference to the NAC model is the same as for BOINC. In contrast to these systems focusing on utilizing available computational resources, the NAC model focuses on using desktop applications for domain specific computation of data for distributed visualization systems.

In addition to the research projects focusing on global computing, there are other projects sharing characteristics with NAC. These include the Scalable Adaptive Graphics Environment (SAGE) [9] based on TeraVision [22] and TeraScope [31], OptiStore [30], Active Data Repository [11], Active Semantic Caching [4], DataCutter [5], ParVox [12], The Remote Interactive Visualization and Analysis System (RIVA) [13], OptiPuter [23], Digital Light Table [10] and Scalable Parallel Visual Networking [7]. However, these systems do not support remote compute resources nor the ability to customize personal computers on-the-fly to become compute nodes in the system.

3 Architecture

The network accessible compute model is realized using a data space architecture, where visualization nodes communicate with compute nodes through a live data set (figure 2). For distributed visualization systems, a separate state server gets user input and provides all visualization clients with the global view state of the visualization through a separate event server. Compute nodes in the system produce data customized to the particular visualization domain of the visualization clients. The network accessible compute resources can be categorized into two classes; static and dynamic. A static

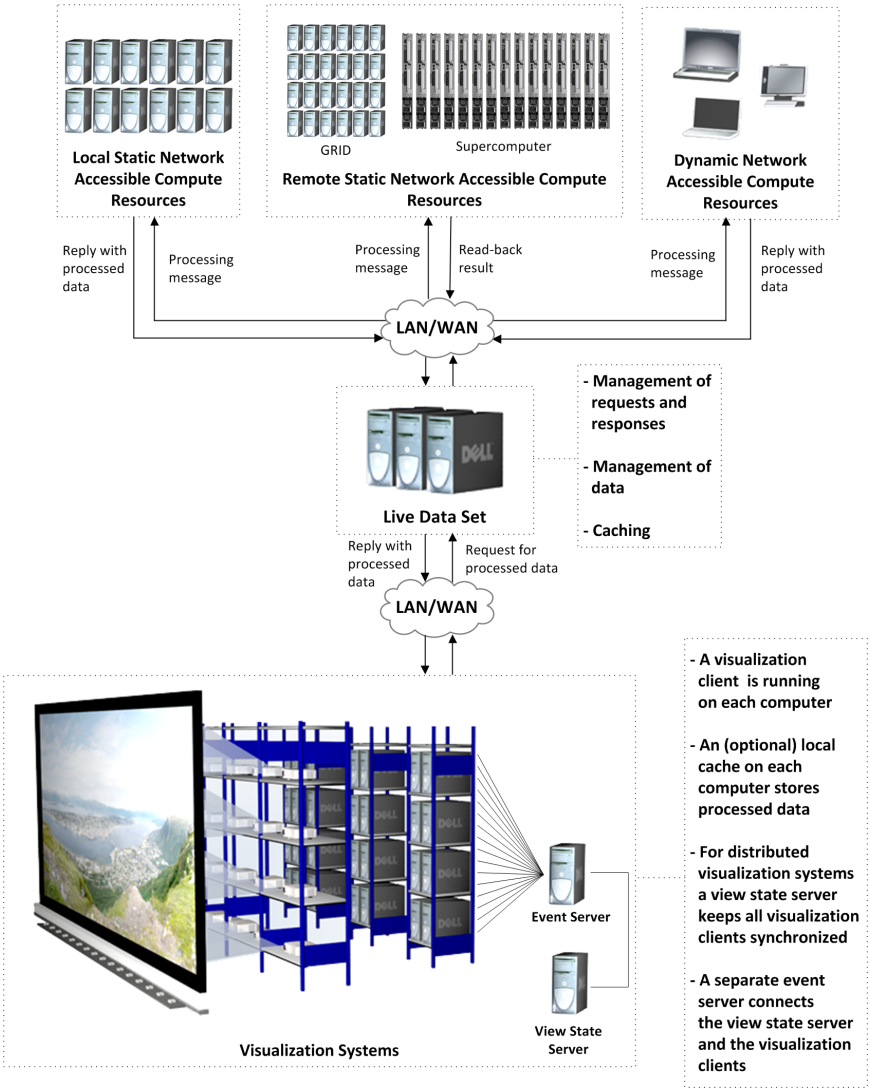

Fig. 2. Architecture.

compute resource is a compute resource that the live data set has been pre-configured to communicate with. This category ranges from clusters of computers to supercomputers which have strict underlying security and access policies (software running on a supercomputer is often prohibited to make outgoing connections). A dynamic compute resource is a compute resource that is customized by the live data set to produce data for the system. A computer can become a dynamic compute node in the system by registering with the live data set to become customized, and then provide information about the type of requests that it can process and what data it will share with the system.

The display-side of the system can query the live data set to get information about all the data that it comprises. This data is a combination of all the data the compute resources can process on behalf of the display-side. From the visualization nodes point of view, the live data set contains all the data pre-processed. However, the live data set will only actually contain data that has been processed, and all requests for data that has not been processed will be sent to a compute node that can produce this data. This is done transparently to the display-side, and thus hides all computation to the visualization clients.

4 Design

The display-side of the system comprises a set of visualization clients that request data from the live data set, which they then use as part of the rendering. Each visualization client combines the view state received from the view state server (via the event server) with the location in the display grid to determine what data to request from the live data set. The live data set contains data customized for the particular visualization domain of the visualization clients, for example maps rendered into image tiles or files converted to a format that the visualization clients understand.

The customization of the dynamic compute nodes is done by the live data set. A computer that wishes to become a compute node in the system initiates contact with the live data set. The live data set responds with a piece of code that is downloaded to the compute node. This code is responsible for performing the initial setup with the live data set. When the compute node has downloaded and started the code, a plugin validation phase is started. The downloaded code contains a set of plugins that can be used to compute processed data. These plugins might be available for different operating systems and installed software in general. Plugins are run in a separate address spaces to utilize multiple CPU cores and support non-thread-safe APIs. Based on the type of plugins the compute node supports, a list of supported data types is generated and sent to the live data set, which then stores information about the compute node and its associated data types for future requests from visualization clients.

Visualization clients can browse the live data set to determine which processed data it contains. For example, if a visualization client contains functionality for processing images but lack the functionality for reading PDF documents, it can request tiled images of the document from the live data set. The live data set sends compute messages to compute nodes that hold the document with an associated plugin. The compute nodes produce image tiles from the document which is returned to the live data set and in turn back to the visualization clients. All data is cached in the live data set to avoid re-computation of data.

5 Implementation

Currently, one visualization system has been implemented as part of WallScope. The system is implemented in C++ using OpenGL for rendering. C++ was chosen for allowing optimization of performance critical parts of the rendering engine. OpenGL was chosen for cross-platform utilization of available graphics hardware. The visualization

system supports gigapixel images, virtual globes, 3D models of various formats and regular images. The visualization system queries the live data set at regular intervals to get an updated list of the processed data that it contains. Most of the data in the live data set can be rendered using different levels of detail. Therefore the visualization system is built to show a visualization of all the data that the live data set contains, enabling users to get an overview, at the same time being able to zoom in at the finest level of detail for each of the data set elements. A user can navigate in the visualization using a touch-free interface constructed from a set of floor mounted cameras. The touch-free interface supports common gestures found in regular touch displays such as panning and zooming. Additionally, the touch-free interface supports 3D touch input allowing users to easily navigate in 3D visualizations.

The live data set is implemented in C++ using Squid [29] as the front-end for caching. The visualization clients request data from the live data set using HTTP. Although HTTP is a relatively heavyweight protocol, especially for usage in high performance distributed and parallel systems, it was chosen for the large number of existing compatible systems (such as the aformentioned Squid cache system) and other applications and utilities (such as web browsers etc.) that can be used to debug and solve problems with the system. Previously performed requests are handled by the Squid cache. If the data for a request is not cached or has expired, the live data set inspects the request, performs a lookup in the compute node list to find a compute node that can process the request, and then sends a message to this compute resource to get processed data.

The live data set has a Java JAR file that contains the code needed for a dynamic compute resource to communicate with the live data set, as well as all the plugins developed for the system. A compute node downloads and executes this JAR file using the Java Network Launch Protocol (JNLP) [26]. The user initiates the customization of the compute node by clicking on a link in a browser. Once downloaded and started, the Java code will validate the plugins to find the compatible plugins for the installed software environment. The plugins are executable files created for a specific software platform. Some of the plugins implemented are plugins to compute processed data from DOC, DOCX, XLS, XLSX, PPT, PPTX, PDF and various 3D formats. These plugins utilize the desktop applications already installed on the computer, for example using Microsoft Component Object Model (COM) [18] to orchestrate a document conversion to a format that can be processed by the NAC service. The processed data ranges from image tiles and PDFs to 3D models that the visualization clients can load. Since the dynamic resources initiates contact with the live data set, the compute resources are available to the system even though they might be behind Network Address Translation (NAT) or a firewall.

6 Experiments

To evaluate the system, four experiments were conducted with the purpose of documenting the performance of the system, and to find potential bottlenecks. In all experiments a 350-page PDF document was used, and the time to rasterize (compute-side) and display (display-side) the document was measured. In experiment one and two, the

number of compute nodes was varied between 1 and 28, and the image tiles produced were encoded using PNG and JPG, respectively. In experiment three and four the produced image tiles (PNG and JPG) were loaded from the live data set's cache and from a local cache on each node.

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 live data set, the event server and the state server (same specifications as the display cluster nodes); and (iii) 28 compute nodes (Intel Xeon Processor E5520 8M Cache 2.26 GHz, 2.5 GB RAM, 4 cores, Hyper-Threading and running the 32-bit version of CentOS release 5.5). Compute nodes were group-wise connected to gigabit switches (6 compute nodes per group). These switches were connected to a gigabit switch connected to a router providing the link to the display nodes. The shared theoretical bandwidth between compute nodes and display nodes was 1 gigabit per second.

For all experiments, the time used to compute and render a 350-page document was measured, with the purpose of identifying the speedup when adding compute nodes to the system, as well as documenting potential bottlenecks. The PDF document was rasterized into image tiles on the compute-side. Each tile had a size of 512x512 pixels and every page of the document comprised six such tiles. This yields a total resolution of 550 megapixels for the 350 pages. In experiment one, PNG was used as the image tile format. In experiment two, JPG was used. For both experiments, 1, 2, 4, 8, 16, and 28 compute nodes were used to compute the result. Each of these nodes had 4 compute processes running to utilize all the cores (not including Hyper-Threading). Every display node was configured to perform 4 simultaneous requests to the live data set. These requests were load-balanced on the available compute nodes by the live data set. In experiment three and four, image tiles were loaded from the live data set's cache and from the local cache on each display node, with the purpose of documenting potential bottlenecks in the cache system and the network bandwidth between the live data set and the display nodes. For these experiments, the same image tiles requested in the previous experiments were used. The number of requests generated for each experiment was 2432. This number is larger than the number of tiles that comprised the document, and is caused by some of the image tiles overlapping between displays and thus are requested at least 2 times. (The image tiles overlapping between display corners are requested by 4 display nodes).

6.2 Results

Figures 3 and 4 show the time and speedup factor for experiment one and two. This includes the rasterization of the document into image tiles on the compute-side including the time to encode the images to PNG or JPG, the transfer of these image tiles from the compute-side, through the live data set, to the display-side, and the loading and rendering of the image tiles on the display nodes.

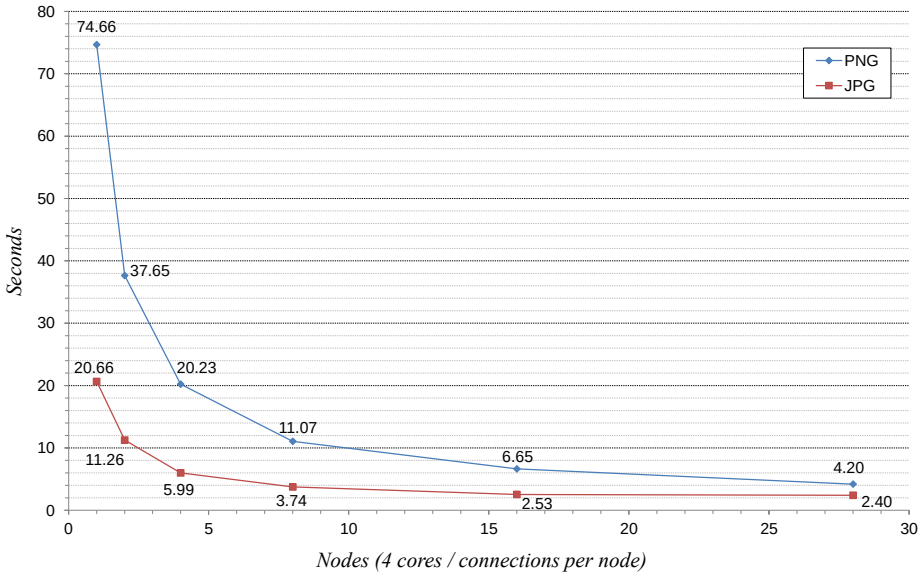

Fig. 3. Time to request and simultaneously display 2432 JPG or PNG encoded image tiles computed from a 350-page PDF document residing at the compute-side. (Compute nodes are increased from 1 to 28).

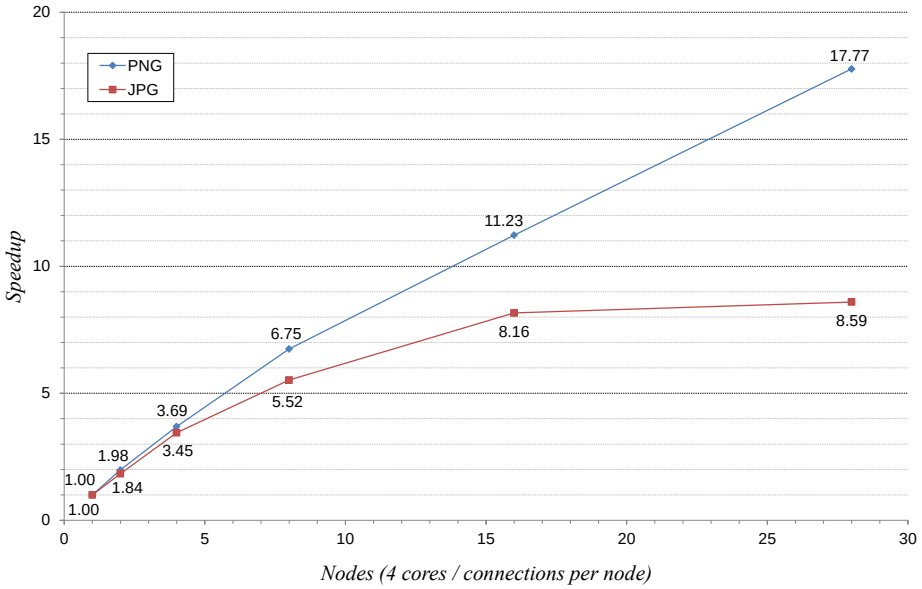

Fig. 4. Speedup factor when requesting and simultaneously displaying 2432 JPG or PNG encoded image tiles computed from a 350-page PDF document when going from 1 to 28 compute nodes.

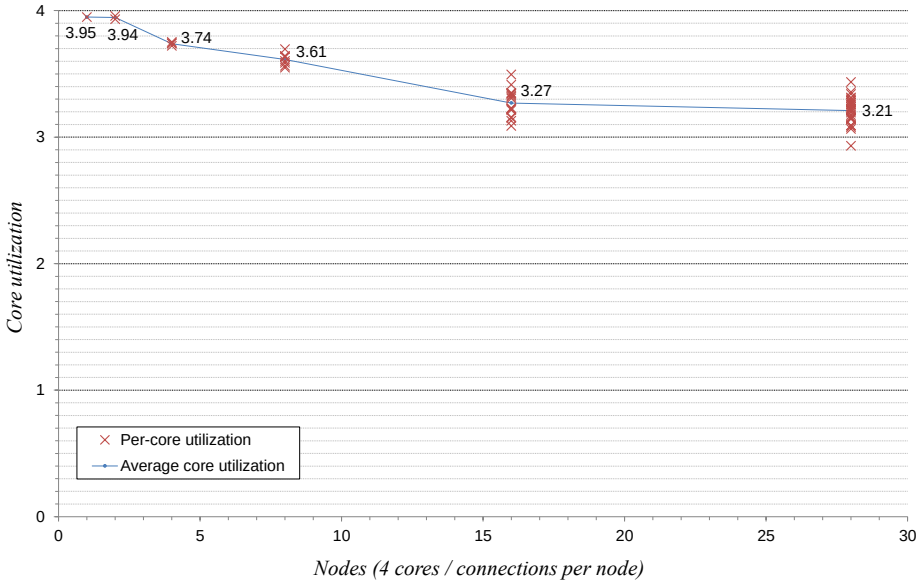

Fig. 5. Compute core utilization when rasterizing the 350-page PDF document to PNG images.

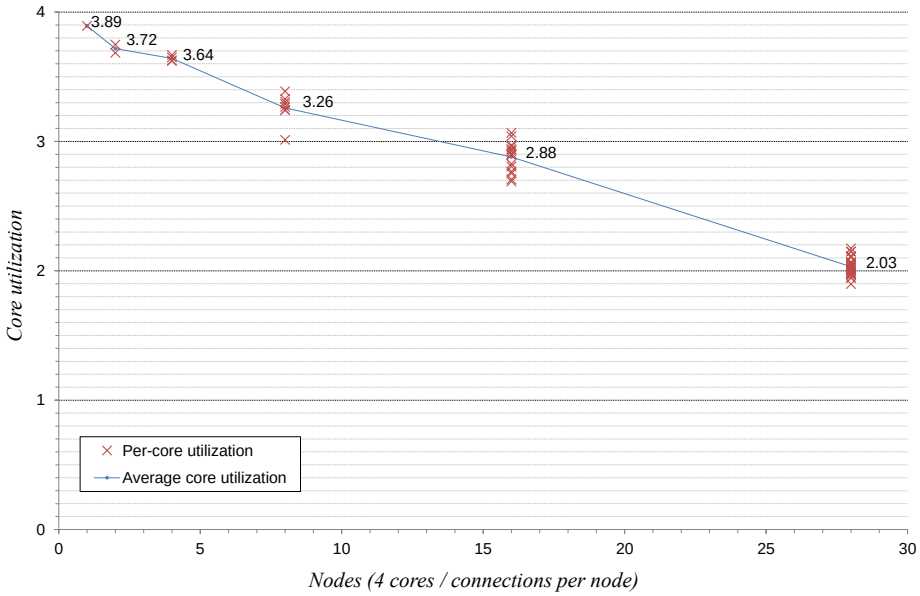

Fig. 6. Compute core utilization when rasterizing the 350-page PDF document to JPG images.

Figures 5 and 6 show the per-core and average core utilization for experiment one and two.

Table 1 shows the average latency for one request in the system when using all 28 compute nodes. Table 2 shows the result of experiment three and four. The load time for the LDS cache includes the time used to request data from the cache, the transfer of the images over the network and the local time used to decode and render the images. The load time for the local cache includes the time used to request the tiles from the local cache, including the time to decode the images and render them.

Table 1. Average latency for a request to complete when using 28 compute nodes

Image Type	Display-Side	LDS	Compute-Side
PNG	0.1521 sec	0.1456 sec	0.1445 sec
JPG	0.0865 sec	0.0574 sec	0.0533 sec

Table 2. Time to request and simultaneously display 2432 PNG or JPG encoded image tiles requested from the live data set's cache or from the local cache on each visualization node

Image Type	Load Time LDS Cache	Load Time Local Cache
PNG	1.694 sec	0.908 sec
JPG	1.305 sec	0.923 sec

6.3 Discussion

As can be seen from figures 3 and 4, the system benefits from increased number of compute nodes. When using PNG as the image format, the total load time is 74.66 seconds using 1 compute node. When using all nodes this time is reduced to 4.20 seconds, which translates to a speedup of 17.77. When using JPG as the image format the time to load the entire document using one compute node is 20.66 seconds. This time is reduced to 2.4 seconds using all compute nodes, translating to a speedup of 8.59. However, as both figures show, the load time and speedup factor does not translate with a linear one to one factor with the use of additional compute nodes. In addition, for JPG the speedup is approximately half of the speedup of PNG when using all nodes and only increases with 0.43 when going from 16 to 28 nodes. This indicates a bottleneck in the system. When the produced image tiles are located in the live data set's cache, the time used to load and display the document on the display-side is 1.694 for PNG and 1.305 for JPG (table 2). The reason that the compute system cannot produce data with this rate is a combination of the latency introduced by computing the image tiles on the compute-side and the number of connections that is established from every node on the display-side. During the experiments every display node had 4 request connections. For PNG the average latency per request is 0.1521 seconds. When using 4 connections this translates to a total average of 3.3 seconds per display node (((2432 / 28) / 4) x 0.1521). However, the compute nodes are idle some of this time. For JPG this is even

worse. The latency per request is 0.0865 seconds, giving a total latency of 1.88 seconds. However, the compute nodes are using 0.0533 seconds per compute request, giving a larger idle time. The result of this can be seen from figures 5 and 6. The core utilization decreases as the number of nodes increases. When using PNG as the image format, the CPU core utilization is 3.95 using 4 cores on one node. This value is reduced to 3.21 using all nodes. For JPG, the CPU core utilization using 1 node is 3.89, which is reduced to 2.03 using all nodes. To solve this situation the display-side could be configured to use more than 4 connections to the live data set. However, there is a tradeoff between the number of connections established from the display-side and the performance of the rendering engine. Request threads are responsible for decoding the data to the rendering engine. Decoding of images is CPU bound and request threads will therefore compete with the rendering engine for CPU cycles on a single-core computer. This will in turn affect the framerate of the visualization. However, this problem can be solved in several ways (separate request functionality from decoding functionally, pipeline requests or use multi-core computers on the display-side with a dedicated core for the rendering engine). In addition to the display-side modifications, the connections between the live data set and the compute nodes should allow for pipelining of requests to increase the core utilization and mask latency. The presented suggestions are all part of future work and currently being investigated.

7 Conclusion

This paper has presented the Network Accessible Compute (NAC) model and a system, WallScope, adhering the model. The NAC model is realized using a live data set architecture, which separates compute nodes from visualization nodes using a data set containing data customized for the particular visualization domain. Visualization clients request data from the live data set, which forwards these requests to available network accessible compute resources. Network accessible compute resources start the relevant personal desktop applications and use them to produce output that can be transferred into display-side compatible formats by the NAC service. The results are returned to the visualization clients for rendering.

Experiments conducted show that the compute resources in the system can be utilized in parallel to increase the overall performance of the system, improving the load time of a PDF document from 74.7 to 4.2 seconds (PNG) and 20.7 to 2.4 seconds (JPG) when going from 1 to 28 compute nodes. This shows that the application output from personal desktop computers can be made interoperable with high-resolution tiled display walls, with good performance and without being limited to the resolution of the local desktop and display. The main bottleneck in the system is the compute-side of the system combined with the synchronous communication mechanism used throughout the system. Currently, work is being done to improve on this. The experiments conducted have shown promising possibilities for displaying of static content such as document and images. Future research will focus on more dynamic content such as collaborative edited documents and videos, in addition to integrating more applications with the compute-side software.

8 Acknowledgements

The authors would like to thank Lars Ailo Bongo, Bård Fjukstad, Phuong Hoai Ha and Tore Larsen for discussions. In addition, the authors would like to thank the technical staff at the Computer Science Department, University of Tromsø, especially Jon Ivar Kristiansen for providing great support on the compute nodes used in the experiments. 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 Computers, Storage and Display Devices, and No. 155550/420 - Display Wall with Compute Cluster.

References

1. N. Abdennadher and R. Boesch. Towards a peer-to-peer platform for high performance computing. In *HPCASIA '05: Proceedings of the Eighth International Conference on High-Performance Computing in Asia-Pacific Region*, page 354, Washington, DC, USA, 2005. IEEE Computer Society.
2. D. P. Anderson. Boinc: A system for public-resource computing and storage. In *5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10, 2004.
3. D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@home: an experiment in public-resource computing. *Commun. ACM*, 45(11):56–61, 2002.
4. H. Andrade, T. Kurc, A. Sussman, and J. Saltz. Active semantic caching to optimize multi-dimensional data analysis in parallel and distributed environments. *Parallel Comput.*, 33(7-8):497–520, 2007.
5. M. D. Beynon, T. Kurc, U. Çatalyürek, C. Chang, A. Sussman, and J. Saltz. Distributed processing of very large datasets with datacutter. *Clusters and computational grids for scientific computing*, 27(11):1457–1478, 2001.
6. G. F. Cecile, G. Fedak, C. Germain, and V. Neri. Xtremweb : A generic global computing system. In *In Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID01)*, pages 582–587, 2001.
7. W. T. Correa, J. T. Klosowski, C. J. Morris, and T. M. Jackmann. SPVN: a new application framework for interactive visualization of large datasets. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, page 6, 2007.
8. T.-M. S. Hagen, D. Stødle, and O. Anshus. On-demand high-performance visualization of spatial data on high-resolution tiled display walls. In *Proceedings of the International Conference on Information Visualization Theory and Applications*, pages 112–119, 2010.
9. B. Jeong, L. Renambot, R. Jagodic, R. Singh, J. Aguilera, A. Johnson, and J. Leigh. High-performance dynamic graphics streaming for scalable adaptive graphics environment. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 108, 2006.
10. D. Katz, A. Bergou, G. Berriman, G. Block, J. Collier, D. Curkendall, J. Good, L. Husman, J. Jacob, A. Laity, P. Li, C. Miller, T. Prince, H. Siegel, and R. Williams. Accessing and visualizing scientific spatiotemporal data. In *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*, pages 107–110, June 2004.
11. T. Kurc, U. Çatalyürek, C. Chang, A. Sussman, and J. Saltz. Visualization of large data sets with the active data repository. *IEEE Comput. Graph. Appl.*, 21(4):24–33, 2001.
12. P. Li. Supercomputing visualization for earth science datasets. In *Proceedings of 2002 NASA Earth Science Technology Conference*, 2002.

13. P. Li, W. H. Duquette, and D. W. Curkendall. Riva: A versatile parallel rendering system for interactive scientific visualization. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):186–201, 1996.
14. M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.
15. Y. Liu and O. J. Anshus. Improving the performance of vnc for high-resolution display walls. In *Proceedings of the 2009 International Symposium on Collaborative Technologies and Systems*, pages 376–383, Washington, DC, USA, 2009. IEEE Computer Society.
16. J. L. Manferdelli, N. K. Govindaraju, and C. Crall. Challenges and opportunities in many-core computing. *Proceedings of the IEEE*, 96(5):808–815, May 2008.
17. Microsoft. `http://msdn.microsoft.com/en-us/library/aa383015(VS.85).aspx`.
18. Microsoft. `http://www.microsoft.com/com/default.mspx`.
19. NVIDIA. `http://www.nvidia.com/object/product_geforce_gtx_480_us.html`.
20. V. S. Pande, I. Baker, J. Chapman, S. P. Elmer, S. Khaliq, S. M. Larson, Y. M. Rhee, M. R. Shirts, C. D. Snow, E. J. Sorin, and B. Zagrovic. Atomistic protein folding simulations on the submillisecond time scale using worldwide distributed computing. *Biopolymers*, 68(1):91–109, 2003.
21. Predictor@home. `http://predictor.scripps.edu`.
22. R. Singh, B. Jeong, L. Renambot, A. Johnson, and J. Leigh. 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, 2004.
23. L. L. Smarr, A. A. Chien, T. DeFanti, J. Leigh, and P. M. Papadopoulos. The optiputer. *Commun. ACM*, 46(11):58–67, 2003.
24. A. C. Sodan, J. Machina, A. Deshmeh, K. Macnaughton, and B. Esbaugh. Parallelism via multithreaded and multicore cpus. *Computer*, 43:24–32, March 2010.
25. D. Stainforth, J. Kettleborough, A. Martin, A. Simpson, R. Gillis, A. Akkas, R. Gault, M. Collins, D. Gavaghan, and M. Allen. Climateprediction.net: Design principles for public-resource modeling research. In *In 14th IASTED International Conference Parallel and Distributed Computing and Systems*, pages 32–38, 2002.
26. Sun Microsystems. `http://www.jcp.org/aboutJava/communityprocess/first/jsr056/jnlp-1.0-proposed-final-draft.pdf`.
27. Tilera. `http://www.tilera.com/pdf/PB025_TILE-Gx_Processor_A_v3.pdf`.
28. B. Vinter. The architecture of the minimum intrusion grid (mig). In J. F. Broenink, H. W. Roebbers, J. P. E. Sunter, P. H. Welch, and D. C. Wood, editors, *CPA*, volume 63 of *Concurrent Systems Engineering Series*, pages 189–201. IOS Press, 2005.
29. D. Wessels, K. Claffy, and H.-W. Braun. 1995. NLANR prototype Web caching system, `http://ircache.nlaur.net/`.
30. C. Zhang. *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, 2008.
31. C. Zhang, J. Leigh, T. A. DeFanti, M. Mazzucco, and R. Grossman. Terascope: distributed visual data mining of terascale data sets over photonic networks. *Future Gener. Comput. Syst.*, 19(6):935–943, 2003.