2008

A Distributed System for Parallel Simulations

Mengxia Zhu
Southern Illinois University Carbondale, mzhu@cs.siu.edu

Nanda K. Yadav
Southern Illinois University Carbondale

A Distributed System for Parallel Simulations

Mengxia Zhu* and Nanda K. Yadav
Computer Science Department
Southern Illinois University
Carbondale, IL 62901
*corresponding author Email: mzhu,nanday@cs.siu.edu

***Abstract*—We presented the technologies and algorithms to build a web-based visualization and steering system to monitor the dynamics of remote parallel simulations executed on a Linux Cluster. The polynomial time based algorithm to optimally utilize distributed computing resources over a network to achieve maximum frame-rate was also proposed. Keeping up with the advancements in modern web technologies, we have developed an Ajax-based web frontend which allows users to remotely access and control ongoing computations via a web browser facilitated by visual feedbacks in real-time. Experimental results are also given from sample runs mapped to distributed computing nodes and initiated by users at different geographical locations. Our preliminary results on frame-rates illustrated that system performance was affected by network conditions of the chosen mapping loop including available network bandwidth and computing capacities. The underlying programming framework of our system supports mixed-programming mode and is flexible to integrate most serial or parallel simulation code written in different programming languages such as Fortran, C and Java.**

I. Introduction

While visualization of 3D volumetric data is a powerful tool to analyze and capture hidden knowledge embedded in raw scientific data, the advantages are better reflected by real-time simulation applications that have monitoring and steering capabilities than applications dealing with preexisting datasets. Current steering systems include SCIRun[1], CUMULVS[2], VASE[3], Progress[4] and RealityGrid[5] that have found wide applications in various scientific fields. However, there are still some challenges that we seek to address: First of all, most of these systems require the installation of various 3rd party libraries such as Globus, SOAP, PVM [6] and AVS [7] and it is usually tedious to set up and configure those packages, which consequently impose a high learning curve for users. Secondly, systems introduce separate visualization packages, which compromise the system's integrity to seamlessly combine various components for efficient computing performance. Thirdly, most of these systems exploit a static system configuration in terms of nodes and network links utilization. Such static mapping strategy is simple and easy to implement but could incur tremendous delay for some applications whose characteristics require a totally different configuration mode.

In order to tackle these challenges, we built a web-based visualization and steering system that can support various types of large-scale simulations executed at remote supercomputers or cluster systems. One of the primary goals of our system is to provide a web interface with functionalities to monitor

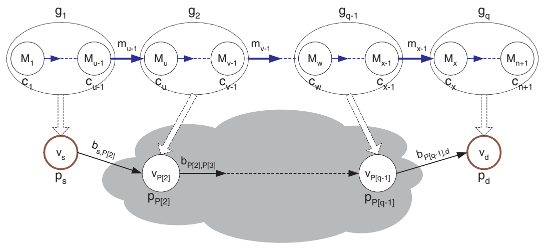


Fig. 1. Mathematical model for pipeline partitioning and network mapping.

and steer an ongoing simulation in real-time. Stray simulations can be terminated or corrected to save computing cycles. The user only needs to open a web browser to fully access the system after proper identification check. Such web front-end completely eliminates the need for any third party softwares or libraries installation by users and makes the system highly accessible. Furthermore, the steering functionality proves to be crucial in saving unnecessary time spent on bad or stray simulations. Our system also adopts the concept of Service Oriented Architecture (SOA) to create a robust, flexible and scalable computing environment, because various computing modules or components coordinate with each other by providing a particular service responding to certain service request. Most importantly, for simulation tasks with linear computing pipelines, we develop and apply polynomial-time optimization algorithms to choose the optimal network loop to achieve maximum frame rate for streaming applications. Simulation results based on distributed nodes are provided to illustrate the importance of our algorithm.

Our system involves many aspects of modern computing techniques such as parallel and distributed computing, network communications, advanced hierarchical data managements, scientific visualization, and rich client AJAX (Asynchronous JavaScript and XML) web design etc. The Ajax web technology allows partial web page update without reloading the whole page to improve the responsiveness at the user end. Parallelism of computation is achieved by executing MPI simulation commands at our Linux cluster machines. Pthreads is utilized to support process parallelism and multiple-client requests at each computing node. Visualization of 3D volumetric datasets are carried out by our own C++ code built upon an open source visualization packages - Visualization Cook Book (VCB) [8].

II. Optimal Pipeline Mapping

A. Analytical model

We now describe an analytical model for the general problem of computing pipeline decomposition and its network mapping as in Fig. 1[9].

The computation pipeline consists of $n+1$ sequential modules, $M_1, M_2, \dots, M_{u-1}, M_u, \dots, M_{v-1}$, ..., $M_w, \dots, M_{x-1}, M_x, \dots, M_{n+1}$, where M_1 is a data source. Module $M_j, j = 2, \dots, n+1$performs a computational task of complexity c_j on data of size m_{j-1} received from module M_{j-1} and generates data of size m_j, which is then sent over the network link to module M_{j+1} for further processing.

An underlying transport network consists of $k+1$ geographically distributed computing nodes denoted by $v_1, v_2, \dots, v_k, v_{k+1}$. Node $v_i, i = 1, 2, \dots, k, k+1$ has a normalized computing power p_i[1] and is connected to its neighbor node $v_j, j = 1, 2, \dots, k, k+1, j \neq i$ with a network link $L_{i,j}$ of bandwidth $b_{i,j}$ and minimum link delay $d_{i,j}$. The minimum link delay is mostly contributed by the link propagation and queuing delay, and is in general much smaller than the bandwidth-constrained delay of transmitting a large message of size m given by $m/b_{i,j}$. The transport network is represented by a graph $G = (V, E), |V| = k+1$, where V denotes the set of nodes (vertices) and E denotes the set of links (edges). The transport network may or may not be a complete graph, depending on whether the node deployment environment is the Internet or a dedicated network.

We consider a path P of q nodes from a source node v_s to a destination node v_d in the transport network, where $q \in [2, min(k+1, n+1)]$ and path P consists of nodes $v_{P[1]} = v_s, v_{P[2]}, \dots, v_{P[q-1]}, v_{P[q]} = v_d$. The pipeline is decomposed into q visualization groups denoted by $g_1, g_2, \dots, g_{q-1}, g_q$, which are mapped one-to-one onto the q nodes on transport path P. The data flow between two adjacent groups is the one produced by the last module in the upstream group; We have $m(g_1) = m_{u-1}, m(g_2) = m_{v-1}, \dots, m(g_{q-1}) = m_{x-1}$. The client residing on the last node v_d sends control messages such as simulation parameters, filter types, visualization modes, and view parameters to one or more preceding visualization groups to support interactive operations. However, since the size of control messages is typically in the order of bytes or kilobytes, which is often much smaller than the size of visualization data, its transport time is assumed to be negligible.

B. Maximum frame-rate

In distributed computation applications producing streaming data such as animations with a number of time steps, processing and visualization data of the same modality is continuously generated, manipulated, and rendered in a pipelined manner. The maximum frame rate that a pipelining system can achieve is limited by the slowest (bottleneck) transport link or computing node along the pipeline. Our goal is to maximize the frame rate by minimizing the time incurred on a bottleneck link/node for applications with streaming data.

Let $1/T^j(v_i)$ denote the maximal frame rate with the first j messages (namely the first $j+1$ visualization modules) mapped to a path from source node v_s and node v_i in an arbitrary computer network. Let $S^j(v_i)$represent the sum of the message sizes of all modules on node v_i with the first j messages mapped from node v_s to v_i. We have the following recursion leading to $T^n(v_d)$:

$$\underset{j=2\ to\ n, v_i \in V}{T^j(v_i)} = \min \begin{cases} \max\left(T^{j-1}(v_i), \left(S^{j-1}(v_i) + c_{j+1} m_j\right) / p_{v_i}\right), \\ \min\limits_{u \in adj(v_i)} \left(\max\left(T^{j-1}(u), c_{j+1} m_j / p_{v_i}, m_j / b_{u, v_i}\right)\right) \end{cases} \tag{1}$$

with the base conditions computed as:

$$\underset{v_i \in V, and\ v_i \neq v_s}{T^1(v_i)} = \begin{cases} \max\left(c_2 m_1 / p_{v_i}, m_1 / b_{v_s, v_i}\right), & \forall e_{v_s, v_i} \in E \\ \infty, & otherwise \end{cases}$$

and $T^t(v_s) = \sum_{i=1}^{t} \left(c_{i+1} m_i / p_{v_s}\right),\ t = 1, 2, \dots, n.$

At each step of the recursion, $T^j(v_i)$ takes the minimal of two subcases. In the first subcase, we do not map the last message m_j to any network link; instead we directly place the last module $M_j + 1$ at node v_i itself. In the second subcase, the last message m_j is mapped to one of the incident network links from its neighbor nodes to node v_i. With the dynamic network configuration capability supported by dynamic performance estimation modules, it is guaranteed that the system will always choose the optimal network mapping scheme resulting in best possible frame-rate on the client side. Dynamic network configuration is essential for today's scientific computing needs where different applications may have tremendously different computing complexities which need to be handled by the changing network environment in terms of constantly changing bandwidth and available computing resources. We proposed a polynomial-time algorithm based on dynamic programming method to achieve this capability of dynamically configuring computing pipeline over a given computing network. The computational complexity of this core algorithm is $O(n \times |E|)$, which guarantees that our system scales well as the network size increases.

III. System Design

The dynamic programming algorithm can be used appropriately group the pipeline modules and map them onto computer network to achieve the optimal network performance. Estimates on computing and transport time will be collected by statistical approaches and network daemons.

A. Advanced web technology

The term RIA (Rich Internet Application) refers to web applications built to replace traditional desktop applications. We borrow and realize this concept by providing a graphic user interface (GUI) that has widgets to allow any operations that a traditional desktop-based system could do. We choose Google Web Toolkit (GWT)[10] as the Ajax development framework. GWT was released by Google in 2006 and the latest available

[1]For simplicity, we use a normalized quantity to reflect a node's overall computing power without detailing its memory size, processor speed, and presence of co-processors, which may result in different performances for both numeric and graphics computations.

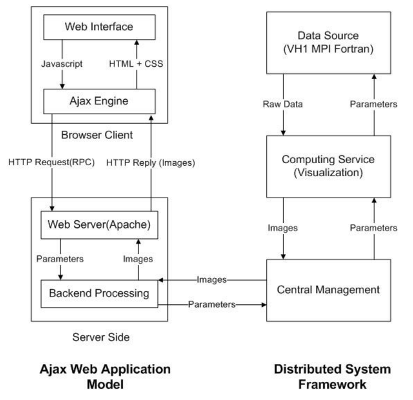


Fig. 2. Implementation components diagram for Ajax Web Technology

stable version is 1.4. Gmail and Google maps are all successful Ajax applications developed and marketed by Google Inc,. GWT compiler can translate java code into Javascript, XML and HTML code enabling users to create nice web applications by writing pure java code. GWT also provides many widgets which can be used to create visually pleasing customized widgets. Some other open-source alternatives for creating Ajax-based applications are Dojo Toolkit [11], Yahoo Ajax library [12], and ASP.NET Ajax from Microsoft [13]. The backend computing of GWT is driven by Remote Procedure Call (RPC), where services of user verification, parameters submission, images receiving, and error handling are all handled through RPC mechanism initiated at the browser end and executed at GWT server end. Building a web-based computing system to replace a traditional desktop-based system provides great flexibility and convenience to users, as people can access the system from anywhere at anytime without installing any software packages except web browser.

Fig.2 shows a implementation components diagram of GWT. As discussed earlier, the client interface, a Ajax-based GUI, allows users to enter steering and visualization parameters and see the final images rendered from simulation datasets generated at remote cluster machines. Ajax model is different from traditional web-application model in such a way that it has an intermediate module called Ajax engine sitting between client and server. The Ajax engine provides asynchronous communication between two ends and coordinates partial page refresh, which only loads changing components and keep the other web components unchanged.

B. System Modules

The system contains several inter-connected modules and all of them will be discussed in detail. Our system contains the following four modules or nodes. 1. Client Module, 2. Central Management (CM), 3. Data Source (DS) and 4. Computing Service (CS).

Central Management: Central Management (CM) node plays the role of central director, as it gathers information about the user requests, dispatch steering parameters to simulation center, and estimate current network conditions when a new steering request is submitted by a remote user etc. CM uses collected information to compute the optimal mapping of computing pipeline and establish a routing table to set up the communication loop. The quit command from the user will cause the CM to propagate quit message and tear down the established communication sockets.

Client module: Client module is responsible for providing a graphical user interface that allows users to make new steering requests. The GUI contains an image widget to displays sequences of rendered images computed by upstream computing nodes. GWT framework and Ajax programming technique are utilized. Ajax has gained rapid popularity as a novel technology to build dynamic asynchronous web content that is both rich in content and faster in page loads. We chose Google Web Toolkit(GWT) along with Remote Procedure Calls(RPC) mechanism of Java to build web client of the system.

Data Source: Data Source is the node that keeps generating simulated scientific raw datasets upon which the users can perform steering and visualization operations. Data source can also be used to house pre-generated data for simple remote visualization purpose when users are only interested in visualizing a static dataset. For streaming applications, DS is usually chosen to be a Linux cluster or a Supercomputer to carry out computing-intensive simulations with parallel capabilities.

Computing Service: Computing Service nodes are mainly responsible for processing, visualization and rendering tasks. For visualization modules, we support surface extraction for volume rendering, raycasting, Linear Integral Convolution(LIC) and streamline methods etc. CS can be a single workstation or a cluster machine. Multiple CS nodes can coordinate with each other to compute various visualization modules and form a streaming pipeline.

C. Simulation codes integration

Our system serves as a universal placeholder for simulation code because it can accommodate different simulation codes written in various languages including C, C++, Fortran, Java or combination of those with minor adaptation. A simulation program usually involves an iterative loop which produces a sequence of 3D datasets at different time steps. Such 3D raw datasets will be processed in a streaming fashion to produce an animation at the browser end. we do not want the steering process to hinder the on-going simulation. The computation going on in the server should be uninterrupted and be asynchronous. To achieve this goal, we strategically insert C function calls at appropriate locations within the simulation code to deal with the two-way communication and passively intercept the steering requests. Fig. 3 is the pseudocode that illustrated the enhanced Virginia Hydrodynamics code [14] written in MPI-based Fortran 90 with inserted C function calls developed separately. For example, PushDataToVizNode will transport the raw dataset from simulation node to processing node; ReceiveHandleMessage will intercept and handle various requests

Virginia Hydrodynamics MPI (VH1) Pseudocode

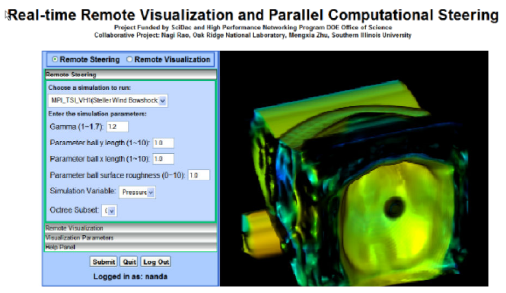
```
MPI_INIT()  // Start MPI enviroment
StartSimulationServer();
AcceptWaitConnection();

 do ReceiveHandleMessage();
 while ( no new Steering request )

  Begin by reading input deck
  Check that arrays are large enough for desired number of zones
  Initialize variables for the new problem
  Restart from old dump files to save time
  Increment dump file name
  // Main Computational Loop
  do
      sweepx;
                MPI_ALLTOALL();
      sweepy;
                MPI_ALLTOALL();
      sweepz;
                MPI_ALLTOALL();
       if MPI_RANK = = 0
          PushDatatoVizNode();
          ReceiveHandleMessage();
          if ( Message is new simulation params )
            UpdateSimulationParameters();
          endif
                MPI_BARRIER()
                Mergevh1data() //To merge all output from slave nodes
       endif
 while ( cycles != endcycles )
 //End of Main computational Loop

 if MPI_RANK == 0
    Write to a file for data needed for post processing
 endif
 MPI_FINALIZE()  //End MPI Environment
```

Fig. 3. System function calls inserted in MPI VH1 code.



Real-time Remote Visualization and Parallel Computational Steering

Project Funded by SciDac and High Performance Networking Program DOE Office of Science

Collaborative Project: Nagi Rao, Oak Ridge National Laboratory, Mengxia Zhu, Southern Illinois University

[✓ Remote Steering] Remote Visualization

Remote Steering

Choose a simulation to run: MPI_TSI_VH1(Steller Wind Bowshock

Enter the simulation parameters:

Gamma (1~1.7): 1.2

Parameter ball y length (1~10): 1.0

Parameter ball x length (1~10): 1.0

Parameter ball surface roughness (0~10): 1.0

Simulation Variable: Pressure

Octree Subset: (

Logged in as: nanda

Fig. 4. Web user interface with a Raycasting image of stellar wind problem.

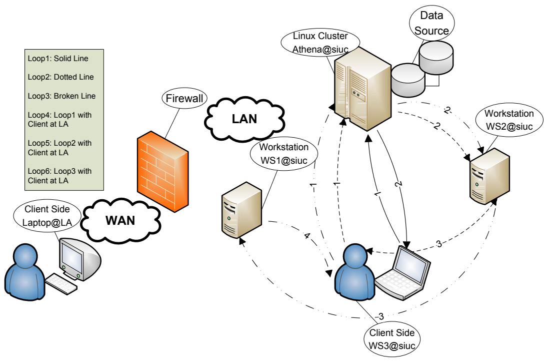


Fig. 5. Six network loops consisting of different nodes, and the number denotes the link index for a particular loop.

such as new steering, quit, rotate and zoom requests, submitted by users; and UpdateSimulationParameters will overwrite the current simulation parameters for the simulation codes with value intercepted by ReceiveHandleMessage.

IV. Experimental results

A. Parallel hydrodynamics simulations

Virginia Hydrodynamics(VH1) is a computational hydrodynamics simulation code developed in 1990 [14]. It is written in Fortran and the latest available parallel version is in Fortran 90 with MPI support. VH1 can model several different hydrodynamics problems like radiation cooling, formation of shocks in tube and gravitational collapse of interstellar cloud. In our system, we have incorporated the MPI version of VH1 and modified the original code to meet the requirements of the system. We have automated the merging process for multivariate datasets, thus the multiple data slices generated by independent processors will be combined into a single one at the end of each step. The merged dataset in netCDF[15] format will be filtered to extract a particular variable of interest and converted to a binary file to save storage space. We run tests on one subproblems of MPI-VH1, Stellar Wind problem. The computing modules of the linear pipeline are mapped to geographically distributed computer nodes to compare the achievable frame rates.

Stellar Wind Problem This sub-problem is designed to simulate the formation of bowshock around a star in a interstellar medium. The parameter to be steered in this case is "gamma", also known as adiabatic index of gas. In this case, we need to control the adiabatic index of hydrogen. To avoid unsteady condition, gam should avoid values less than 1. We used bounds checking at the web interface to prevent users from entering invalid gamma values. We applied iso-surface extraction and ray casting visualization techniques to expose region of interest within the datasets. Fig. 4 showed a screen shot of a rendering image using ray casting technique.

B. Framerates with different mapping schemes

We wish to demonstrate that the optimal pipeline partition and mapping scheme chosen by our system outperforms all other alternatives in terms of frame-rates. A MPI Simulation on Stellar Wind problem using 8 cluster nodes with 16 processors is used with ray casting visualization technique. We set up a testbed consisting of five nodes(three workstation and one Linux cluster named as Athena housed in Faner Hall at SIUC; one computer located at Los Angeles) across the WAN and LAN to validate our algorithm and system framework as shown in Fig. 5. Because our Linux cluster machine running the MPI simulations and other workstations are protected behind SIUC firewall without port exception, all distributed computing nodes are chosen to be located within SIUC campus except the web client node for convenience purpose. The experimental results and conclusions drawn from LAN tests can be applied to the scenario of WAN because of the similar underlying assumptions. For all mapping schemes, the simulation nodes acting as DS as well as the web server is mapped to the Athena Linux cluster. The CS nodes and CM nodes are mapped to different Linux workstations of WS1 and

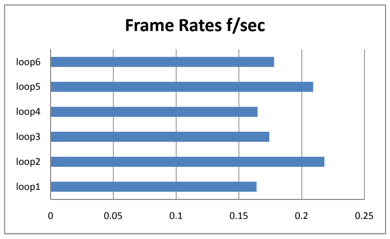


Fig. 6. The frame-rates achieved with different network loops

WS2 depending on the specific loops. The web client nodes are either selected to be a computer in Los Angles or the workstation of WS3 at SIUC.

We collected experimental results by running the system on six different pipeline mapping loops for the same hydrodynamics problems and visualization technique. In Fig. 5, the first loop maps CM, DS and CS nodes to athena, and client resides on WS3. The second loop maps CM and CS to WS1, and client is on WS3. The third loop maps CM to WS1, CS to WS2, and client is at WS3. The fourth loop is the same as the first loop except the different client side at LA. The fifth loop is the same as the second loop except the LA client. The sixth loop is the same as the third loop except LA client. We measured and compared the different frame-rates achieved at the web client side by alternative pipeline loops in Fig. 6. We observed that the optimal pipeline, namely loop 2 achieves the highest frame-rate due to the minimum bottleneck computing time along that network path. Our experiments further illustrate that different pipeline mapping schemes will lead to considerably different streaming performance. Such difference will be much bigger if applications are conducted across WAN with unstable network conditions. Thus, choosing the optimal network configuration in a dynamic way is important to achieve interactivity with satisfactory network performance. It is also interesting to note that that different client sides did not cause noticeable disparities on the frame-rate. For example, frame rates of loop 1 is similar to that of loop 4. It indicates that the computing bottleneck is not on the transport link from web server to client end. However, we experienced different delays for the first image from loops with different client sides.

V. Conclusion and Future Work

We have presented the design and implementation issues of a distributed web-based system for monitoring and steering of large-scale simulation codes. Interactive steering has replaced traditional "batch" simulations to boost the resource efficiency and user interactivity. Although many current steering systems have been developed to provide monitoring and steering capabilities for large-scale scientific simulations, there are still some drawbacks and deficiencies that needed to be addressed.

We designed and implemented a fully functional web-based monitoring and steering system that can be easily accessed by users without installing third party packages. Such real-time monitoring and steering functionality enables users to "see" ongoing simulations on the fly. Users can track the evolving dynamics of the simulation and can choose to either correct the stray simulation or terminate bad simulation to save computing cycle and time. We incorporated several sub-problems of hydrodynamics applications into our system to demonstrate the interactive steering feature of the system as well as the responsive rich web interface built using Google web toolkit based on Ajax techniques.

Our future work include the development and deployment of automated network daemon across WAN and dedicated network to measure and report the network conditions such as available bandwidth and computing cycles to be used as input for our dynamic programming algorithm. It would allow our system to dynamically configure itself to achieve the best possible performance. We also aim to provide more functionality at the web interface. For example, some statistical analysis tools can be added to analyze the data from many aspects. In addition, we will look into ways to generate a movie from history images to allow users to play back for detailed examinations. In addition, we also plan to apply these heuristics to the mapping problems for Directed Acyclic Graph(DAG) structured workflows. The most effective solution will be integrated into the implementation of the future system framework.

VI. Acknowledgement

This project is supported by Oak Ridge National Laboratory under contract number 6400006042-2224068. Authors would like to express gratitude to N. S.V. Rao and Q. Wu for their insightful advices, J. M. Blondin for providing MPI VH1 code and L. Zhu for help with conducting remote client tests.

References

[1] S. Parker and C. Johnson, "Scirun: A scientific programming environment for computational steering," in *The Proceedings of Supercomputing Conference*, 1995.
[2] G. G. II, J. Kohl, and P. Papadopoulos, "Cumulvs: Providing fault tolerance, visualization, and steering of parallel applications," *The International Journal of Supercomputer Applications and High Performance Computing*, pp. 224–235, 1997.
[3] "Common data format," http://nssdc.gsfc.nasa.gov/cdf.
[4] J. Vetter and K. Schwan, "Progress: A toolkit for interactive program steering," in *Proceedings of the 1995 International Conference on Parallel Processing*, 1995, pp. 139–142.
[5] B. Boghosian, P. V. Coveney, S. Dong, L. Finn, S. Jha, G. Karniadakis, and N. Karonis, "Nektar, spice and vortonics: Using federated grids for large scale scientific applications," *Journal of Cluster Computing*, vol. 10(3), pp. 351–364, 2007.
[6] "Pvm," http://www.csm.ornl.gov/pvm.
[7] "Avs," http://www.avs.com.
[8] "Visualization cook book," seelab, University of Tennessee.
[9] M. Zhu, Q. Wu, N. Rao, and S. Iyengar, "Optimal pipeline decomposition and adaptive network mapping to support distributed remote visualization," *Journal of Parallel and Distributed Computing*, vol. 67, pp. 947–956, 2007.
[10] "Google web toolkit," http://code.google.com/webtoolkit.
[11] "Dojo toolkit," http://dojotoolkit.org.
[12] "Yahoo ajax library," http://developer.yahoo.com/yui.
[13] "Asp.net ajax framework," http://ajax.asp.net.
[14] "Virginia hydrodynamics," http://wonka.physics.ncsu.edu/pub/VH-1.
[15] "Network common data format," unidata.