Visualization of Distributed Processes Using “Data Jewelry Box” Algorithm

Yumi Yamaguchi Takayuki Itoh
IBM Research, Tokyo Research Laboratory
{yyumi, itot}@trl.ibm.com

Abstract

Visualization of distributed processes is useful for the management of large-scale distributed computing systems. Reactivity and scalability are especially important requirements for such visualization of distributed processes. We have proposed the visualization technique "Data Jewelry Box" algorithm, which satisfies both of above requirements. The technique can be applied for the visualization of distributed processes, however, the algorithm has a problem that may yield much different data layouts even among very similar datasets. This is a serious issue for the seamless visualization of time-varying data.

To solve the problem, we propose the extension of "Data Jewelry Box" algorithm. The extension places data elements referring positions of the previous data layout, so that the extension can yield similar layouts among similar datasets. This paper introduces the extended algorithm, and proposes the visualization system for distributed computing systems using the extended algorithm.

1. Introduction

Visualization of distributed computing systems is important for the purposes of management and planning of the systems. Actually, some monitoring systems for distributed environments, including OpenView by Hewlett Packard [12], became popular because of their convenience. Some novel distributed computing architecture, such as the Open Grid Services Architecture (OGSA) [5, 17], will allow users to freely utilize computer resources rather than existing distributed systems, because of the capability of flexible creation of service instances. This means that usage of distributed computing resources may be more complicated when such architectures become more popular. We therefore think that real-time visualization of distributed processes on such architectures will be important for the purpose of management and planning of computer resources.

We assume that satisfaction of the following requirements is desirable for the visualization of distributed processes:

[Requirement 1: Reactivity] Distribution of processes generally changes second-by-second. It is important to very quickly represent the distribution for real-time monitoring of distributed computing systems.

[Requirement 2: Scalability] Parallel application may distribute thousands of processes on large-scale distributed computing systems. To monitor the overall behavior of the parallel computation, it is desirable to represent all of thousands of processes in one display. In other words, it is desirable to reduce the occupancy of display layout areas.

[Requirement 3: Hierarchy] Above thousands of processes can be categorized according to several attributes, such as CPUs, users, and applications. The processes can be therefore arranged as hierarchical data according to the categorization. Hierarchical data visualization techniques are therefore useful for the visualization of distributed processes.

[Requirement 4: Similarity] Hierarchical data formed by above categorization usually slightly vary while a short time step. In other words, two hierarchical data while a short time step are usually very similar. It is desirable that visualization technique generates very similar images between very similar data for the seamless representation of time-sequence of the data.

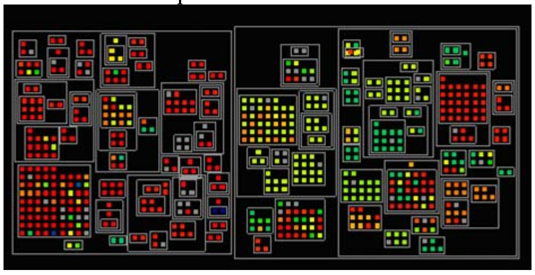

Figure 1 Example of hierarchical data visualization by Data Jewelry Box.

We have proposed Data Jewelry Box [7, 8], a visualization technique for providing overviews of large-scale hierarchical data. As shown in Figure 1, Data Jewelry Box represents hierarchical data as nested rectangles. It represents leaf-nodes as icons, and non-leaf-nodes as nested rectangular borders. The technique satisfies requirements 1, 2, and 3, since it calculates display layout of thousands of nested nodes in one second or so. However, it has a bottleneck for requirement 4, since it sometimes yields very different display layout results among very similar datasets.

This paper proposes an extension of Data Jewelry Box algorithm [19] that improves the similarity among time-series data. Given the display layout of previous data, it calculates positions of icons and rectangular borders in a normalized coordinate. Let us call the table of the positions "template." By referring the template, the extension calculates similar display layout for a similar data. It is especially useful for seamless representation of slightly time-varying hierarchical data.

This paper also presents an implementation of the visualization of distributed processes using the extension of Data Jewelry Box algorithm. The implementation supposes that a client application first has a set of processes, and then sends all the processes to a service working on a server. We call this service the "portal" in this paper. Receiving the set of processes, the parent server creates several service instances and assigns the processes to the instances. After performing the processes, the instances return the results of the processes to the portal. Finally, the portal returns the collection of results to the client application. During this procedure, the portal manages the status of all the processes, and constructs a hierarchical data structure consisting of all of the processes. The implementation frequently updates the hierarchical data, and Data Jewelry Box algorithm frequently renders the data as Scalable Vector Graphics (SVG) image files.

In this paper we used a parallel meshing system as the application of the presented visualization technique. Meshing is a geometric technique that finely divides input geometric regions into small pieces of triangular or quadrilateral elements. Large-scale manufacturing products usually consist of hundreds or thousands of mechanical parts, and it is often necessary to mesh the geometries of such enormous numbers of parts. Parallelization of the meshing process is therefore important in such product manufacturing fields. We implemented the meshing system under OGSA, and monitor the meshing processes by using our visualization technique.

2. Related work

2.1 Distributed processes on Open Grid Services Architecture

Visualization of distributed processes presented in this paper has been implemented on OGSA [5, 17], which is the novel architecture that integrates Grid Computing technologies by combining Web Services mechanisms. Grid Computing is a distributed computing infrastructure that enables transparent access to and sharing of heterogeneous computing resources and software components across distributed users and environments. Reference [4] defined properties of key connectivity, resource, and collective protocols to realize these features of Grid Computing. Web Services is a framework that enables remote and automatic execution of software components via the Internet. OGSA enables the flexible utilization of computing resources of Grid infrastructure by applying interfaces similar to Web Services. Here, OGSA defines the open software components as "Grid service," and transiently working software components as "Grid service instances." OGSA has been proposed to provide mechanisms for registering and discovering Grid Services, and for creating and executing Grid service instances.

A typical scenario using OGSA is as follows: Factory services for creating Grid service instances are first deployed on Web servers. Programs call the factory services to create the instances, and then call functions of the created instances to execute the target services.

The above mechanism allows software developers to flexibly control securities and manage distributed computer resources rather than the earlier Grid technologies. In the other word, it is possible that usage of computer resources and software services will be more complicated, because the service instances can be freely created on arbitrary computer resources. Behavior of systems may often be very different from what system engineers expected, and therefore monitoring of the systems will be important. That is the main reason why we are focusing on the visualization of distributed processes on OGSA.

2.2 Existing hierarchical data visualization methods

Hierarchical data visualization is a hot topic in visualization research, and actually many techniques have been recently presented.

Tree representation is the most popular hierarchical data visualization technique, used in many programs such as famous file system viewers. Several variations, such as the Hyperbolic Tree [11], Cone Tree [3], and Fractal Views [10], have been described for the interactive visualization of large-scale hierarchical data sets. It is also suitable for visualizing the connectivity among nodes. Carriere et al. represented all parts of large-scale hierarchical data by Cone Tree in [3]. It provides a good overview of large-scale data; however, their result contains sparse regions.

Treemap [9], which represents hierarchical data in a manner like nested column charts, is a well-known space-filling approach for hierarchical data visualization. The space-filling approach is especially useful for the representation of statistics, and overview of the data in compact display spaces. However, it has shape-related limitations of leaf-nodes. Squarified Treemap [2] and Ordered Treemap [15] improved the shapes of rectangular subregions, but still they do not guarantee their aspect ratios. Quantum Treemap and Bubblemaps [1] were recently proposed for the layout of icons or thumbnails when those aspect ratios are fixed. Ordered Treemap also claims the seamlessness of the representation time-varying data, however, still it often yields much different layout results among the time-series data in some parts.

Against authors propose techniques for representing hierarchical data as 2D nested structures, but 3D nested structures have been applied in some techniques. Information cubes [13] is a 3D hierarchical data visualization approach, which positions data in nested semitransparent hexahedrons. H-BLOBS [16] is also a 3D hierarchical data visualization approach, positioning data in nested semitransparent isosurfaces.

3. Extension of Data Jewelry Box

This section presents our hierarchical data visualization technique "Data Jewelry Box." First part of the section introduces the overview of "Data Jewelry Box," and then proposes its extension for seamless representation of time-varying data.

3.1 Data Jewelry Box: A hierarchical data visualization technique

Authors have proposed a hierarchical data visualization technique "Data Jewelry Box," especially useful for interactively providing overviews of large-scale data [7, 8]. As shown in Figure 1, our technique represents the hierarchy of input data as nested rectangles. In this figure painted dots denote leaf nodes, and gray rectangular borders denote non-leaf nodes. Bigger borders denote higher-levels of the hierarchy, and smaller ones denote lower-levels. Remark that the name of the technique just denotes the concept, and does not target photo-realistic representation of jewelry box itself.

Figure 2 shows an illustration of the order of data layout in our technique. Our algorithm first packs icons (painted square dots in Figure 2) that denote leaf nodes, and then generates rectangles that enclose the icons to denote non-leaf nodes. Similarly, it packs a set of rectangles that belong to higher levels, and generates the larger rectangles that enclose them. Repeating the process from the lowest level toward the highest level, the algorithm places all of the data onto the layout area.

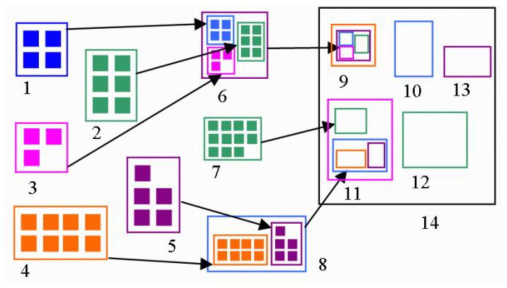

Figure 2 Order of the placing rectangles to represent the hierarchical data. Numbers in this figure denote the order of the process.

Here, one non-leaf-node connects to several children nodes. They are represented as rectangular icons or borders, and should be packed inside a rectangular border. To calculate their positions, the algorithm places a set of rectangles one-by-one, while it satisfies the following two conditions:

[Condition 1] Rectangles must be placed without overlapping each other.

[Condition 2] Rectangles should be placed where the layout area occupied by the rectangles is minimized.

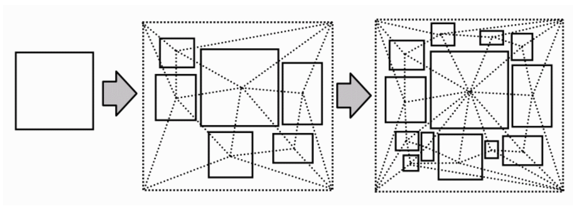

Figure 3 Illustration of the rectangle packing algorithm.

The above placement is somewhat similar to packing problem, which is a famous computational geometry problem, and has been applied to various industry fields, such as VLSI circuit, paper patterns for clothes, and cutting pattern of sheet metals for mechanical parts. Many of these problems apply optimization schemes for the minimization of layout areas; however, our algorithm favors accelerating the rectangle packing process rather than entirely minimizing the layout space. We therefore did not apply optimization schemes to find the configuration of rectangles. Our algorithm sorts the rectangles according to their areas, and places them on a display space in the sorted order, as shown in Figure 3. Here the algorithm used a heuristic to quickly find gaps and place the remaining rectangles in the gaps. The heuristic uses a triangular mesh connecting centers of previously placed rectangles.

The algorithm refers to triangles in the order of their sizes, and generates several sampling points inside the triangles for the trial placement of a rectangle. If the rectangle can be place without overlapping any previously placed rectangles at one of the sampling points and extending the display area, the algorithm decides to place the rectangle there. If there is no point that the rectangle can be placed without extending the display area, the algorithm decides to place the rectangle where the extension of layout area is the smallest and no overlap is happen. References [7, 8] describe the algorithm in detail.

3.2 Proposed extension of Data Jewelry Box for visualization of time-varying data

Figure 4 shows the concept of the extension of Data Jewelry Box algorithm. The extension uses “templates,” which describe positions of nodes of hierarchical data at the previous time step in a normalized coordinate. Given the next hierarchical data, the extension refers the template while it calculates positions of nodes of the data, so that they are placed as similar as the previous data. Positions of previous data are translated into a normalized coordinate so that all positions are within (-1,-1) to (1,1), since the size of layout area usually changes even if the next data is very similar to the previous data.

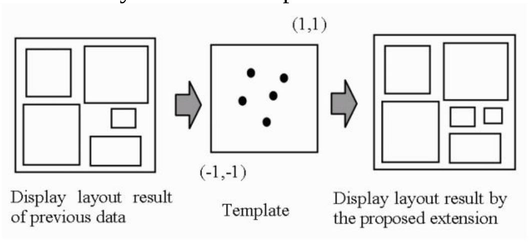
(1,1)
(-1,-1)
Display layout result of previous data
Template
Display layout result by the proposed extension

Figure 4 Concept of the extension of “Data Jewelry Box.”

In addition to two conditions described in Section 3.1, the extension applies one more condition:

[Condition 3] Rectangles should be placed where it is enough close to the position described in the given template.

The extension places rectangles under the three conditions. Similar to the original Data Jewelry Box algorithm, the extension places given rectangles one-by-one. While the trial placement of a rectangle, it refers triangles one-by-one, and generates sampling points inside each triangle. Follows are the procedure of placing rectangles by the extension. Reference [19] describes the algorithm in detail.

3.2.1 Order of placing rectangles. Against original Data Jewelry Box places rectangles in the order of their area, as shown in Figure 5(left), the extension places as the following order:

(1) The largest rectangle is first placed.
(2) Other rectangles are placed one-by-one, in the order of closeness to the largest rectangle, which are calculated from positions described in a template.

The above “adjacency-based order” prevents to yield unnecessary gaps each other.

3.2.2 Order of referring triangles. Against original Data Jewelry Box refers triangles in the order of their area, as shown in Figure 5(right), the extension refers them in the order of distances to the position of currently placing rectangle described in a template. The order accelerates the discovery of adequate position of the rectangle. Our implementation generates sample points on the lines connecting vertices and edges of the triangle, and where currently placing rectangle can touch with previously placed rectangle without yielding unnecessary gaps. Detail is described in reference [19].

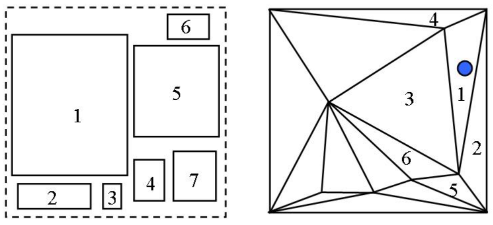

Figure 5 The extended Data Jewelry Box algorithm. (Left) Order of placing rectangles. (Right) Order of referring triangles.

3.2.3 Evaluation of sampling points. While the trial placement of a rectangle at a sampling point, the extension decides that:

(1) If the rectangle overlaps with at least one of previously placed rectangles, the rectangle never be placed there.

(2) Otherwise, the extension calculates the value $aD+bS$, where a and b are user-defined constant values, D is the distance between the sampling point in the normalized coordinate and the position of the rectangle described in the template, and S is the layout area after the placement of the rectangle. After trying the placement with sampling points, the extension decides to place the rectangle where $aD+bS$ is minimum.

Generally $aD+bS$ values increase while referring triangles in the above order. It means that the extension can skip many of distant triangles to refer.

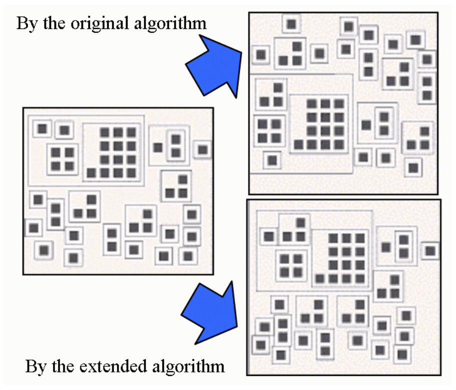

Figure 6 Example of display layout by the original and the extended Data Jewelry Box algorithm. (Left) A result for the data at the previous time step. (Right) Two results for the data at the next time step.

3.2.4 Example. Figure 6 shows the example of hierarchical data visualization by the original and extended Data Jewelry Box algorithm. Left image represents the data at the previous time step, and right two images represent the data at the next time step. The next data is very similar to the previous data, except just two nodes are added from the previous data. The original algorithm calculates the display layout without referring the result for the previous data, and consequently it may yield much different layout though the two data is quite similar. The extended algorithm refers the layout for previous data as a template, and yields very similar layout for the next data.

4. Visualization of distributed processes by extended Data Jewelry Box

This section describes the implementation of and experiments with the visualization of distributed processes. We implemented it on OGSA. First part of the section describes the architecture of our implementation. Next part describes about the parallel meshing system, an application we executed on the architecture. Final part describes some examples and results.

4.1 Architecture of the visualization of distributed processes

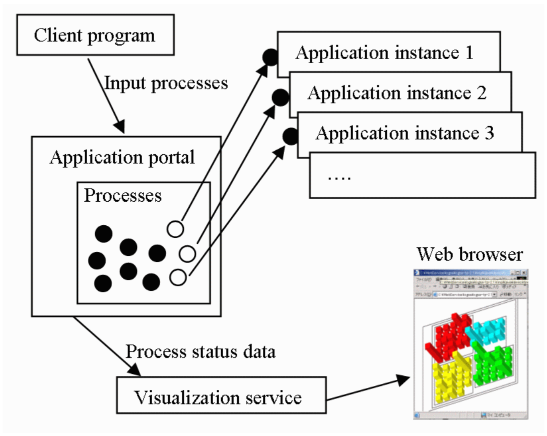

Figure 7 Architecture of visualization of distributed processes on OGSA.

Figure 7 shows the architecture of our implementation. The client program first creates the application portal instance and sends a set of processes to the portal instance. The portal instance then creates several application instances on several servers, and sends the processes to the application instances. The portal instance also creates the visualization instance on a Web server, and sends the URL of the image of the visualization results to the client program. The portal instance manages the status of the processes running the application instances, and frequently sends the status of the processes to the visualization instance. The visualization instance also frequently generates images of the visualization results. Our implementation generates the images in the scalable vector graphics (SVG) format so that users can display them on Web browsers.

The features of the architecture are as follows:

[Simple client programming] In this implementation a client program just sends all of the processes to a single portal instance and receives all the results from the portal instance. It does not need to consider the parallelization.

[Remote process monitoring] Since the visualization instance outputs SVG format images, the distribution and status of the processes can be monitored from anywhere, since the client is reachable by the Web server on which the instance is running.

[Independence of visualization instance] Since the visualization instance is separated from the application instances, it can easily be used for a variety of other applications.

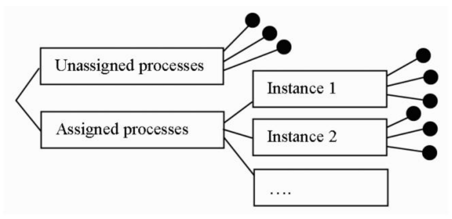

Figure 8 Data structure for the visualization of distributed processes.

Currently, our implementation simply classifies processes into “assigned process” and “unassigned process,” and again classifies the assigned processes according to the instances which processes are assigned. The relationships between the processes have hierarchical form and therefore this data can be the input data for Data Jewelry Box. Figure 8 shows the data structure of processes for the visualization.

4.2 Application: Parallel Meshing System

We applied meshing technique as an application of the visualization of distributed processes. Meshing is a geometric technique to finely divide input geometric regions into small pieces of triangular or quadrilateral elements. It is mainly used for Computer Graphics (CG), Computer Aided Design (CAD), and Computer Aided Engineering (CAE). Figure 9 shows an example of meshes used in CAE.

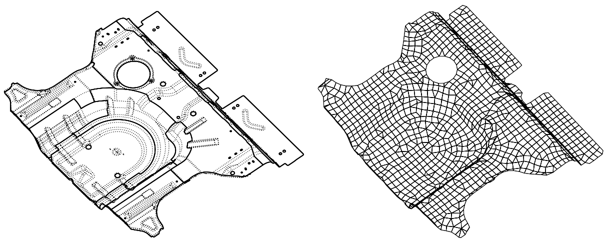

Figure 9 Examples of meshes. (Left) Input geometry. (Right) Output quadrilateral mesh. (copied from reference [6].)

Computational dynamics approach used in CAE is very sensitive to the results of meshing. To obtain adequate solutions in a reasonable computation time, the density and directionality of meshes should be controlled properly, and the shapes of elements should be nearly equi-angle. Due to the difficulties of these requirements, the automation of the meshing process is complicated.

One of the authors has proposed physically-based meshing techniques that automate the meshing process [14, 18]. The techniques have realized nearly-full automation of meshing procedures; however, its computation cost may be a bottleneck. The input geometry shown in Figure 9 consists of hundreds of curved surfaces, and our implementation needs several minutes to generate mesh all the surfaces by a PC with 1GHz CPU. In the design analysis phase for large manufactured products such as automobiles, airplanes, and electric generators hundreds or thousands of mechanical parts are often meshed at the same time. If the above physically-based meshing technique is applied to the CAD models of such large products, it will take about one day to mesh all the mechanical parts using one CPU. Parallelization of the automated meshing techniques is therefore important for time efficient analysis.

We implemented the physically-based meshing technique as application instances shown in Figure 7. In our implementation the client program sends a set of geometric data to the portal service, and receives the meshing results from the service. The portal service sends the geometric data to meshing service instances one-by-one, and receives the meshing results from the instances. Here, we believe that the computation times of meshing parts are estimated by a function of number of vertices, and total computation time is approximately minimized by sending the parts in the order of the estimated computation time of each part. Our implementation roughly forecasts the number of vertices N_v before the meshing process begins. The implementation calculates $N_v = A / l^2$, where A is the area of a part is, and l is the expected length of edges of the output mesh. After sorting all the parts according to N_v , our implementation sends the parts to the meshing instances in the sorted order.

4.3 Example

Figure 10 shows an example of 6 snapshots of the visualization. The example represents processes as bars, and their classifications by gray borders. The client program sends 80 mechanical parts to the portal instance, and the 80 processes are assigned to 4 meshing instances. The current implementation of the visualization instance refreshes the image every 10 seconds. It is possible to refresh the image more frequently, but every 10 seconds is most comfortable on our environment because our current SVG viewer often refreshes the image slowly. It simply colors the bars according to the classifications, and calculates heights according to the computation times. The blue bars at the left denote unassigned processes, and other bars on the right side denote assigned processes. The snapshots show that processes are gradually assigned to the 4 meshing instances. They also represent how many processes each instances has, and how much computation time each process takes. Layouts among the snapshots are very similar by the effort of the extended algorithm.

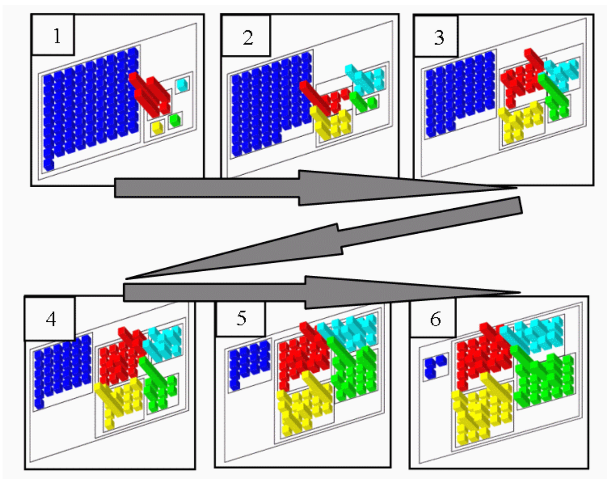

Figure 10 Example of visualization of distributed processes.

4.4 Extension for multiple clients

Section 4.1 described the architecture for the visualization of processes sent from a single client. This section describes the extension of the architecture for the visualization of processes sent from multi clients.

Figure 11 shows the overview of the extended architecture. The scenario of this architecture is as follows: First, clients send requests to the parent portal service with the ID or name of the application they want to run. Receiving a request from a client, the parent portal creates an instance of specified application portal, and returns the URL of the visualization results image to the client. The applications portal then creates several application instances, and distributes the processes into the instance. The application portal also sends the status of their processes to the visualization instance. The visualization instance frequently generates images of the status visualization, and the clients frequently access the images to monitor the processes of the whole distributed computing environment. Also, processes can be categorized according to user-specified attributes, such as users, servers, instances, applications, so the visualization instance can provide various views for various purposes of the process visualization. This extension will help system designers optimize the configurations of distributed systems, and help application users to effectively utilize the computation resources.

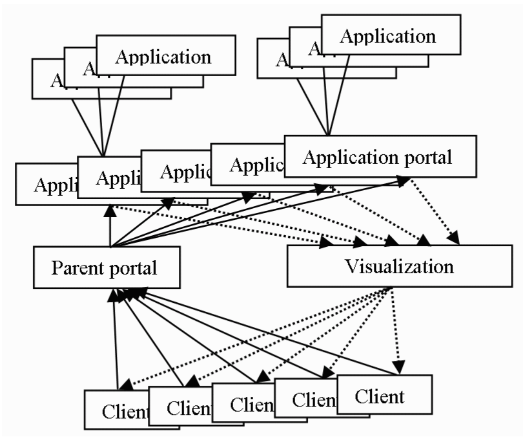

Figure 11 Extended architecture for visualization of processes sent from multi clients.

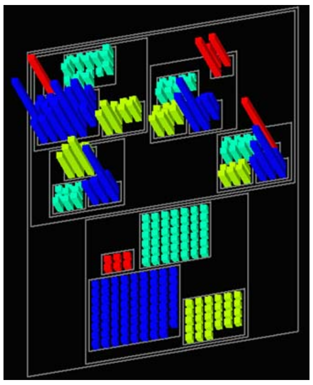

Figure 12 Example of visualization of processes sent from 4 clients.

Figure 12 shows an example of visualization of processes by the extended architecture. In this example 4 clients send sets of processes, and they are distributed to 4 instances. Colors of bars denote clients, and heights of bars denote the computation time. Bars in the lower part of the figure denote unassigned processes. Bars in the upper part are divided into four rectangular regions which represent instances. This figure denotes that the blue client has sent the most number of processes, and the number of remaining processes is also the largest. The red client has sent the least number of processes, but its each process is somewhat heavy. Load balance between 4 instances does not look even, so system designers should think of utilization of computer resources.

5. Conclusion

This paper proposed the extension of Data Jewelry Box algorithm for seamless representation of time-varying hierarchical data. The extension uses “templates,” which describes positions of nodes of hierarchical data at the previous time step. By referring the templates the extension calculates very similar display layout of hierarchical data at the next time step.

This paper also presented architecture of visualization of distributed processes using the extended Data Jewelry Box algorithm. We implemented the architecture on OGSA, and used a parallel meshing as an application of the visualization technique. This paper showed examples of the visualization.

This work is still prototype level, and we are thinking of the following future topics:

[Experiments of scalability] Presented visualization technique can provide a representation of large-scale and deeply-nested hierarchical data that includes thousands of leaf nodes. We would like to confirm the scalability of the visualization with thousands of processes in a real distributed computing environment.

[Improvement of packing algorithm] Minimization of layout areas is important for scalability of the visualization technique, but examples of extended algorithm, shown in Figure 11 and Figure 13, do not look nearly minimize the areas. It is a complicated problem to satisfy both scalability and similarity, and its improvement should be a future discussion.

[Variety of representations] Our implementation simply classifies processes according to instances, colors the bars according to classifications, and calculates heights according to the computation times. However, other kinds of criteria, such as usage of memory and disk space, could be applied to the classification of processes and the calculations of colors and heights. Also, the current implementation represents processes as bars, but other data elements, such as users, CPUs, or disks, could be represented. We would like to study what kinds of representations are useful for the monitoring of distributed computing systems.

References

[1] Bederson B., PhotoMesa: a zoomable image browser using quantum treemaps and bubblemaps, *UIST 2001*, pp. 71-80, 2001.

[2] Bruls D.M., Huizing K., Wijk J. J., Squarified Treemaps, *Data Visualization 2000 (joint Eurographics and IEEE TCVG Symposium on Visualization)*, pp. 33-42, 2000.

[3] Carriere J., Kazman R., Research Paper: Interacting with Huge Hierarchies beyond Cone Trees, *IEEE Information Visualization ’95*, pp. 74-81, 1995.

[4] Foster I., Kesselman C., Tuecke S., The Anatomy of the Grid: Enabling Scalable Virtual Organizations, *International Journal of High Performance Computing Applications*, Vol. 15, No. 3, pp. 200-222, 2001.

[5] Foster I., Kesselman C., Mick J. M., Tuecke S., The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, http://www.globus.org/research/papers.html.

[6] Inoue K., Itoh T., Yamada A., Furuhata T., and Shimada K., Face Clustering of a Large-scale CAD Model for Surface Mesh Generation, *Computer-Aided Design*, Vol. 33, No 3., pp. 251-261, 2001.

[7] Itoh T., Kajinaga Y., Ikehata Y., Yamaguchi Y., Data Jewelry Box: A Graphics Showcase for Large-Scale Hierarchical Data Visualization, *IBM Research, TRL Research Report*, RT0427.

[8] Itoh T., Kajinaga Y., Ikehata Y., Data Jewelry Box: A Graphics Showcase for Large-Scale Hierarchical Data Visualization, *IPSJ Graphics & CAD Research Report*, 2001-CG-104, 2001. (In Japanese)

[9] Johnson B., Shneiderman B., Tree-Maps: A Space Filling Approach to the Visualization of Hierarchical Information Space, *IEEE Visualization ’91*, pp. 275-282, 1991.

[10] Koike H., Fractal Views: A Fractal-Based Method for Controlling Information Display, *ACM Transactions on Information Systems*, 13, 3, pp. 305-323, 1995.

[11] Lamping J., Rao R., Pirolli P.., The Hyperbolic Browser: A Focus+context Technique for Visualizing Large Hierarchies, *Journal of Visual Languages and Computing*, Vol. 7, No. 1, pp. 33-55, 1996.

[12] OpenView, http://www.openview.hp.com/

[13] Rekimoto J., The Information Cube: Using Transparency in 3D Information Visualization, *Third Annual Workshop on Information Technologies & Systems*, pp. 125-132, 1993.

[14] Shimada K., Yamada A., Itoh T., Anisotropic Triangulation of Parametric Surfaces via Close Packing of Ellipses, *International Journal on Computational Geometry & Applications*, Vol. 10, No. 4, pp. 417-440, 2000.

[15] Shneiderman B., Wattenberg M.., Ordered treemap layouts, *IEEE Information Visualization Symposium 2001*, pp. 73-78, 2001.

[16] Sprenger T. C., Brunella R., Gross M. H., H-BLOB: A Hierarchical Visual Clustering Method Using Implicit Surfaces, *IEEE Visualization 2000*, pp. 61-68, 2000.

[17] Tuecke S., Czajkowski K., Foster I., Frey J., Graham S., Kesselman C., Grid Service Specification, http://www.globus.org/research/papers.html.

[18] Viswanath N., Shimada K., Itoh T., Quadrilateral Meshing with Anisotropy and Directionality Control via Close Packing of Rectangular Cells, *9th International Meshing Roundtable*, pp. 227-238, 2000.

[19] Yamaguchi Y., Itoh T., Data Jewelry Box II: A Graphics Showcase for Large-Scale Data Visualization Using Templates of Position Information, *IPSJ Graphics & CAD Research Report*, 2002-CG-108, 2002. (In Japanese)