Using Visualization to Support Network and Application Management in a Data Center

Danyel Fisher, David A. Maltz, Albert Greenberg, Xiaoyu Wang,
Heather Warncke, George Robertson, and Mary Czerwinski

Abstract—**We argue that there is a continuum between completely manual and completely automated management of networks and distributed applications. The ability to visualize the status of the network and applications inside a data center allows human users to rapidly assess the health of the system – quickly identifying problems that span across components and solving problems that challenge fully autonomic and machine-learning based management systems. We present a case study that highlights the requirements on visualization systems designed for the management of networks and applications inside large data centers. We explain the design of the Visual-I system that meets these requirements and the experience of an operations team using Visual-I.**

Index Terms—**Data center, operations, dynamic configuration**

I. INTRODUCTION

The dream of the fully autonomic data center—one that is expressed declaratively, maintained and healed automatically—is still elusive. For any but the simplest data centers, human operators need to periodically intervene: writing maintenance scripts, controlling updates, and modifying systems. Existing tools, such as those springing from a recovery oriented computing perspective, have had positive impact [10] but fall short. Self-repair capabilities have blind spots (e.g., heisenbugs), and encounter errors that they cannot control [13]. Experienced data center operators have found that any autonomic system has the potential to spin out of control— for example, half of all upgrades fail.

Today, when situations arise that autonomic control systems do not handle, such as unexpected faults, application bugs, or emergent behaviors, data center operators are typically forced to revert to the simplest tools—command-line interfaces, point-and-click interfaces that generate long lists of servers, etc.. These primitive tools force humans to manually track and correlate many details: tasks they are not good at performing.

We argue that there is a continuum between fully automated systems and fully manual systems, and that tools for visualization can span the gap between them. For example, visualization tools help operators *understand* the current state of the system, even when the system is in an inconsistent or an unusual state owing to an ongoing upgrade. Visualization can help operators *discover correlated behaviors* that are critical to debugging the system. Finally, visualization tools help *deal with inconsistencies* -- autonomic management tools have difficulty when the structure of the application does not fit the assumptions of the management system. A visualization tool allows operators to see the information they need and then direct the system correctly, without requiring changes to the assumptions of the management system.

In this paper, we present a case study of a data center service that is managed via partial automation. We describe the tools that this service uses, and highlight the challenges facing a visualization system for managing data center services. We then briefly discuss Visual-I, our visualization framework that overcomes these challenges. Finally, we report how Visual-I was applied to the data center application considered, and initial operator experiences with Visual-I.

Our goal was not to build a tool customized for one particular service; rather, it was to build a general tool for data center operations applicable to any cloud service (i.e., a data center application that provides services to users across the Internet). Working with a specific team allowed us to understand how that team creates, maintains, and uses a mental model of their server and network topology. Our results thus far provide a concrete data point on the state of the art in network and distributed system management inside data centers.

II. PROBLEM DOMAIN

Implementing a cloud service involves constructing a single coherent front-end interface out of a complex cluster of routers, switches, servers, and databases. Today, these distributed systems can be composed from on the order of 10,000 physical servers. Given the scale and complexity of the hardware and software, under normal operating conditions hundreds of hardware or software components may be in various degraded states: failing, undergoing upgrade, or failed. Though the larger cloud services build-in replication and resilience to cope with these conditions; nevertheless, things can and do go wrong; for example,

unanticipated dependencies lead to significant service outages. Operators and developers need tools to proactively identify looming problems, to localize and diagnose problems that arise in the field, and to assure unanticipated failures are not triggered during service upgrade.

Today, the operators of these systems have ready access to enormous lists or tree-views of individual components, with a blizzard of configuration and usage data available for viewing behind each component. In addition, new service features, and corresponding new sets of logging features, are born every day. Operators are not lacking in data about their services. However, they are often lacking in the ability to rapidly gain insight (e.g., spot meaningful patterns and outliers) and correlate across the deluge of available configuration, behavior and health data.

We next examine what these challenges mean for a particular cloud service, Microsoft's distributed address book, ABCH, a common service supporting several of Microsoft's Live applications, including Messenger. We focus on tools for the *tier 2* team; i.e., the operations team conversant in the details and theory of operation of the service, and which is responsible for problems that cannot be delegated to automation and to less expert *tier 1* support. Bugs that cannot be fixed by tier 1 or 2 escalate to service developers (tier 3).

A. ABCH: A Typical Cloud Service

The Address Book Clearing House, or ABCH, is a medium-sized cloud service that stores users' address books and presence information. It maintains several hundred back-end databases and around 100 front-end servers that service requests for users' address books from web-based email, instant messaging tools, and other sources.

We met with the operators of ABCH several times on site visits, including two "over-the-shoulder" debugging sessions as they worked through recent crises. We read design documents for their system, and collected replicas of their configuration files and status databases in order to understand how their data was organized. We read email threads and trouble tickets from system failures to understand how they went about solving problems. Finally, we presented prototypes and got feedback to drive our design process.

Like many other cloud services, ABCH has a front-end/back-end architecture, with some added complexity. An external request specifies a given users' address book to be retrieved. The request arrives at one of three "affinity clusters" of front-end server machines. Each cluster has information about one-third of all address book entries. The front-end servers then use a master index to look up which back-end database has the information about this user, and forward the request to the affiliated back-end database. On the back-end, database servers each store a number of databases in a structure of mirrored replicas; servers are organized in "sets" that share backups with each other.

ABCH's topology changes regularly: clusters are re-organized for better balance or to make more space available.

ABCH includes several components that make the application more autonomic. For example, "Webstore" is a database management system that automatically maintains certain desirable topological properties of databases. It responds to both transient and permanent failures by adjusting backup patterns and maintaining the flow of data between primary and secondary databases. Webstore also maintains software state for front-end machines and assists in automatically deploying updates.

B. Tools at ABCH

ABCH, like other services at Microsoft, uses its own set of tools for tracking its status; an additional set of general tools give information about the data center as a whole. The service's specific tools are often customized versions of software designed for enterprise deployment (such as Microsoft Operations Manager and IBM Tivoli). Others are custom tools written locally: "ABCHRunner" tracks end-to-end performance of the system by periodically injecting a series of well-known queries into the front-end and timing the response.

During the over-the-shoulder debugging sessions, however, we found that these tools were a poor fit to the users' mental model. Operators think of the system in terms of connections and clusters, where clusters are groups of machines with similar roles and connections are the relationships between them. However, their tools focused on individual machines—they would consult one custom tool to check a machine's details, switch to another tool for its connections, and query a database for its status. As difficult as it was to track services on one machine, it was even harder to move from one role (e.g., front-end) to another (e.g., load balancer), or to separate out clusters from each other. This caused the team to talk about their systems mainly independently: a discussion of the front-end would have little discussion of the back-end, for instance.

The ABCH team largely uses static views to track system topology. An operator in ABCH draws a diagram in Microsoft Visio periodically. While the diagram shows most of the general features of the system, it cannot keep up with changes. The diagram is obsolete nearly as soon as it is drawn, although it remains sufficiently close to accurate that the team continues to use it as a reference. The team also keeps a series of spreadsheets that list topologies, software versions, etc.. These are updated on a more regular basis, although the team only partly trusts them, as they are populated manually.

In our conversations with the ABCH team, we learned that not only do the monitoring and autonomic management tools have a limited notion of how they are configured, but that the team would strategically *lie* to the tools. For example, a network configuration tracking system was unable to handle the fact that the team maintained a

network configuration that did not match the expected template—some machines used more IP addresses than the configuration tracking system could accommodate. The ABCH team responded by pushing a limited version of the data into the configuration management tool, and then keeping their more complete configuration in spreadsheets and Visio diagrams. While successful as a work-around, it means that some data cannot be automatically retrieved from any system, as it does not exist in machine-readable form.

C. Challenges for Operations Monitoring Tools

Abstracting our observations on ABCH operations, the most important problems with the ecosystem of tools are:

Scalability: Many of the tools built for enterprise applications use mechanisms such as tree-views (akin to expanding/collapsing folders in Windows Explorer) for listing the servers, and these mechanisms become unwieldy with the thousands of servers in cloud services.

Single Perspective: The "tool per component" model forces operators to mentally coalesce and correlate information, resulting in extra steps for the operator as they switch between tools and reducing their ability to achieve situational awareness. Since high-value cloud services are often built by combining other cloud services, this means that the most important services do not have a single tool from which the status of the entire service can be viewed. Instead, the separate tools for each component must be sequentially viewed and the information from them manually integrated and interpreted by the operator.

Abstraction failure: Current tools do not reflect the way operators think about their systems, putting the onus on them to unscramble the component-by-component flat lists of health and status into a meaning hierarchical and structured model of the system.

Topology: Most tools represent servers, switches and routers in lists, and have little or no way of representing or separating elements by network topology or other clusters.

Monitoring overhead: Many tools come with their own monitoring infrastructure. If a single server is to be monitored by several tools, it will often require several monitoring applications installed on it, at substantial cost in processor cycles and network traffic.

Bad/Inconsistent data: Each monitoring tool operates in its own way and has its own idiosyncrasies. Existing tools do not accommodate occasional out of range or non-compliant data. Further, experience shows that sources of meta-data may be inconsistent with each other (e.g., assigning the same server to two different roles or clusters). Today's tools do not help operators cope with, identify, or resolve these discrepancies.

Context loss: Users are forced to switch among multiple tools, and must maintain the context of the jobs they are working on (e.g., the name of the server they are modifying) using ad hoc and error prone methods (e.g., using the copy/paste clipboard).

D. Scenarios

We identified the following three scenarios as most critically in need of visualization support: health monitoring; software upgrades; and crisis resolution.

Health monitoring: Of the tools operators use today, most are request-driven, waiting for an operator to make specific queries; the rest use alarms to alert operators to changes in status. Unfortunately, this structure makes it difficult to monitor the health of systems smoothly. Operators are unlikely to notice a problem until something goes very wrong, setting off an alarm; fluctuations that involve some servers running abnormally—but not bad enough to be at alarm levels—will not be noticed at all. A visualization that allows operations staff to know at a glance how the system is operating would help anticipate problems early.

Software update: No matter how well tested, deploying a software update is a fragile, multi-step process: each server must be taken off active status, updated, and brought back on-line. In the process, a far-away service may suddenly discover that it depends on a now-disabled function, or long-dormant bugs may come to the surface. Operations teams monitor upgrades carefully - continually making decisions whether to continue the upgrade process. Monitoring the progress of an update and tracking its effects on the rest of a service is critical to correctly making these decisions.

Crisis resolution: When a true crisis occurs, tier 2 operations staff use their extensive knowledge of the system configuration to try to figure out what factor is at fault. Visualization can help surface regularities in the failure, and give the operations team a fast way to examine their active data.

III. RELATED WORK

There are large bodies of previous work that attempt to reduce or eliminate human involvement in running distributed applications and networks inside data centers. For example, autonomic computing uses algorithms to monitor and manage systems, and the Map/Reduce paradigm restructures computations into a form that can be more easily managed autonomically. Our work is synergistic to these, providing a more powerful way for operators to understand and interact with their systems and providing a fallback when these techniques fail.

Previous visualization research on data centers, cloud services, and server farms has focused on network security. The annual VizSec workshop (see vizsec.org) collects a variety of solutions for data security.

Other visualization tools can be found in distributed computing research [5][6]. The "Ping Radar," one component of a distributed system health monitoring suite [2] is a special-purpose visualization of system state for distributed systems; it is based on a general data collection platform.

MAYA [1], a tool suite for operators at Amazon, shares some concepts with Visual-I. MAYA also represents
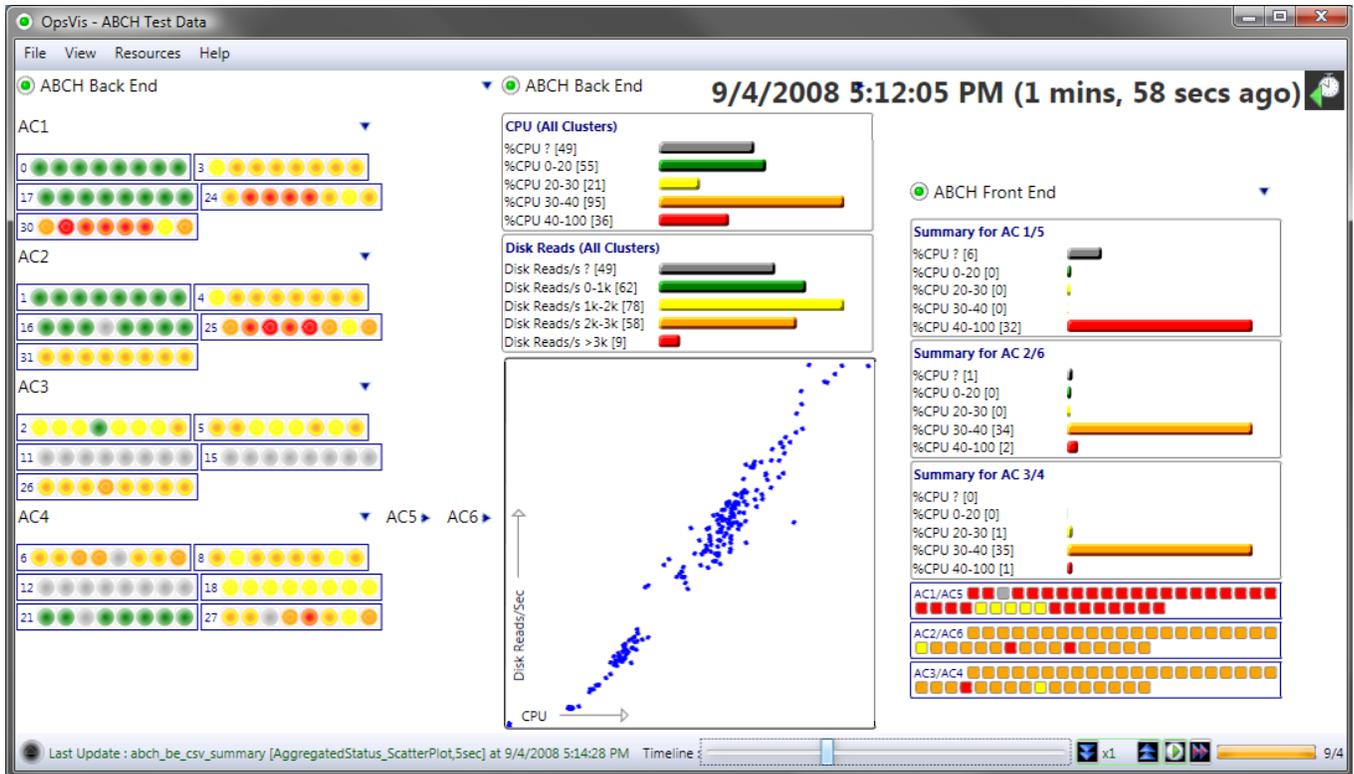
Figure 1. A portion of a Visual-I display in operation at ABCH. Three different front-end clusters each communicate with a pair of back-end database clusters. Each database cluster consists of 5-6 databases sets; each set contains eight servers, each hosting four primary and four secondary databases. Color of a server indicates CPU and disk utilization. Summaries (e.g., histograms of load status) provide awareness when detailed views are closed, and scatter plots identify aberrant outliers. Time controls allow recent history to be replayed as a "weather-loop" easing correlation of behavior across time. The color-map has been changed for this publication; in use, the background is black for better readability in the low-light of the operations center.

network connections between machines, although it uses a node-link diagram to represent machine clusters. MAYA's goal is to automatically detect anomalies and to cluster alerts and alarms.

A substantial thread of research has applied various visualization techniques to massive scale networks [8][9]: for subsets of the internet [7][4], peer-to-peer architectures [10], and others. These systems focus on visualizing large Internet structures. Visual-I deals with large structures, but also visualizes dependencies and richer information about each node.

IV. SYSTEM DESIGN

A. Visual-I Interface

Visual-I provides a single view of the entire cloud service, as a network. It shows high-level objects, such as clusters of servers, as a single unit. For example, the three front end clusters are depicted on the right, along with associated CPU load histograms. These units can be interconnected with edges that visualize dependencies between the clusters and the network configuration. Each composite object can be expanded to see the next level down: sets of machines, or individual servers and databases. Figure 1 shows Visual-I in action as configured for ABCH. Bubbles and small rectangles represent individual machines and databases, while top-level boxes represent logical clusters of machines.

This layout was developed through several iterations with the ABCH team. On the right, boxes represent clusters of stateless front-end servers. Each cluster is expanded to show boxes representing individual servers. Each back-end cluster is divided into five or six sets; each set has eight servers responsible for a total of 32 databases. The rectangles and their annotations make all of this clear at a glance (without mental gymnastics).

Visual-I is designed to ease the visualization of logical structures, like clusters, with information overlays that leverage the structure to help users correlate data. Overlaid on the ABCH display are CPU and disk IO utilization represented as color. The visualization makes immediately clear the difference in loading between the three front-end clusters, as well the missing data from three back-end database sets.

In Visual-I we have sought visual cues that allow the eye to correlate across multiple objects in the same role, as well as dependent objects in different roles. Scatter-plots allow operators to identify servers with high CPU not due to disk activity. Brushing out regions in the scatter-plot highlights the servers wherever else they appear; this allows operators

to interact with the display and look for correlation between plotted variables and the role of the server or its position in the topology.

The images are animated, with decorations and displays such as CPU load and activity information updating as the data changes. Visual-I also provides a time-loop feature so that past behavior can be replayed at high-speed (similar to weather map animations). This helps operators correlate the overlaid data across time to discover recurring and time-dependent behaviors.

B. System Components

A Visual-I configuration is described in four sections:
1) The underlying datasources that must be accessed to collect topology information (that is, which machines will be visualized and their relationships to each other).
2) The models that interpret the datasource in the user interface.
3) The status datasource that monitors the values for the machines.
4) The visualization mappings that translate status into visual clues (color, shape, and so on).

Each of these sections is specified in a declarative file that configures the visualization. This declarative form makes it easy for users to modify and customize their visualization and to share it with others.

The configuration file is written in XML. The config file represents an object hierarchy directly, making it easy for users to interpret the configurations they have generated. We found that operators are comfortable writing SQL queries to retrieve topology and monitoring data from databases, so the configuration is organized around SQL queries that return rows indicating how servers and routers should be grouped into topologies or clusters, and SQL queries providing the data that will be visualized over the resulting topology. A sample XML file that generates two front-ends, with each server colored by CPU load, is in Figure 2.

Given the scale of the systems involved, users cannot look at all servers all the time. Accordingly, each visualizer within Visual-I provides a method for summarizing its data. The most useful summarizers we have built so far include: (1) simply shrinking the display area allocated to the visualization; (2) providing a histogram of data values, with shadows [12] used to indicate temporal change; (3) allowing data streams to provide an aggregation function (e.g., max, min, average).

Visual-I avoids increasing monitoring overhead by reusing pre-existing data sources, rather than requiring new monitoring agents. The system is flexible to data of variable type and quality (databases, spreadsheets, and glue logic are welcome) the system shows whatever sources it is directed to, and it ignores data sources that are not operative.

Operators configure the visualization by writing or updating the XMLconfig file. While a networking operator

```xml
<OpsVisRoot>
  <TopoSQLDatasource Name="LoadFrontEnd"
  Connection="Server=SQLUTIL;Database=OpsVis;" >
    <TopoSQLDatasource.Query>
      SELECT AC AS KeyColumn, FrontEnd
      FROM BackendInfrastructure
      WHERE AC = '{0}'
    </TopoSQLDatasource.Query>

<StatusSQLDatasource Name="MachineCPUTimes"
  ApplyTo="AC1FrontEnd, AC2FrontEnd""
  RefreshFrequency="10" DataType="Value"
  MappingFrom="CPUTIME" MappingTo="Background"
  MapConverter="ValueToBrightness"
  Connection="Server=SAMPLERFARM;Database=Samples;"
  AggregateWith="Avg" MinValue="25" MaxValue="35">
    <StatusSQLDatasource.Query>
      SELECT Computer.Name AS KeyColumn,
        snd.SampleValue AS CPUTime
      FROM SampledNumericData snd
      WHERE SampleType='CPU'
    </StatusSQLDatasource.Query>
  </StatusSQLDatasource>

<OpsVisRoot.ModelCollection>
    <FrontEndModel Name="AC1FrontEnd"
TopoDatasource="LoadFrontEnd"  Param="AC1"/>
    <FrontEndModel Name="AC2FrontEnd"
TopoDatasource="LoadFrontEnd"  Param="AC2"/>
  </OpsVisRoot.ModelCollection>
</OpsVisRoot>
```

Figure 2. A simple Visual-I configuration file that visualizes two front-end clusters, and colors them by CPU status information.

might choose to have a configuration that most prominently displays network information, for instance, a database operator might suppress network information entirely for information about their area of specialty. Tools to help construct the XML remain future work, but in our experience operators are quite comfortable working directly with XML.

Adding new data to the system requires only writing an SQL query and copying a few parameters; as such, Visusal-I users have found it straightforward to modify the visualization to their needs.

V. DEPLOYMENT EXPERIENCE REPORTS

In this section, we highlight salient experiences—and changes in direction--that came out of design sessions with the ABCH team.

Our first major change came from learning that the team 'lied' to its databases (see above). We needed to adjust the system so that diverse data sources could be integrated to obtain for topology and usage data.

Second, the ABCH team had certain 'expected' values of data; they wanted to highlight with colors values that were out of these ranges. Thus, for example, any CPU value above 60% should be colored 'too high'; color gradations should be used for CPU values between 25 and 50. When the color scale was re-adjusted, the team was able to catch times when multiple machines were surging.
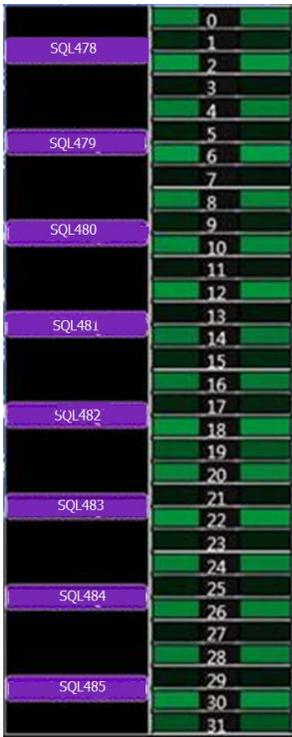
Figure 3. Detail of Visual-I. A set of databases (in green) and servers (in purple). Brighter databases have more rows. The green stripes are a result of a backup cycle as data is moved from one set of databases to another.

A. Feedback and Validation

ABCH's operators were quick to adopt and customize Visual-I to their application. The team used the configuration mechanism to customize data sources and views in order to produce a new system-wide view specific to their needs. Visual-I's color gradation design enabled the ABCH team to easily understand a node's health relative to other servers in the same cluster. This provides a huge leg up in reacting to problems and localizing their root cause. The ABCH tuned their data sources to focus on server processing outliers identified in Visual-I.

In running Visual-I, the ABCH team found meaningful patterns not easy to detect in the original tools. For instance, the visualizer for database sets (Figure 3) shows server name on the left as a horizontal bar; on the right are the databases themselves—each with its number (0-31) and two boxes where color intensity indicates activity of the "primary" and "secondary" copy of the DB. Operators were pleased to discover Visual-I directly depicted activity characteristic of the system's backup procedures. During the backup process, the contents of some databases are merged together, leaving others emptier. This creates a distinctive alternating pattern of bright and dull green stripes that stand out in views of the database clusters. This new information assists monitoring performance and correctness.

VI. CONCLUSION

Cloud services are designed and implemented as networks of distributed systems. The underlying distributed systems are characterized by rapid change (in infrastructure, software, and workload), and by use of replication of components (such as servers, data bases, and switches) as the key to scaling out to meet demand in a high performance and reliable manner. Monitoring these systems effectively and economically is a major challenge. Subtle problems inevitably arise in cloud services that impact user perceived performance, and that, unfortunately, are hard to detect, localize, and diagnose.

Visual-I helps to meet these challenges by enabling developers and operators to create visualizations that provide insight at a glance into anomalies and variability across the systems. It provides the ability to create perspectives that match the way cloud service developers and operators think about their systems.

Visual-I demonstrates how visualization can be used to provide rapid understanding of data, and to reveal new phenomena that would be hard to detect, diagnose or repair via autonomic tools. Our approach leverages the human visual system's ability to correlate patterns and deal with inconsistencies to help operators succeed in situations where autonomic tools fall short.

REFERENCES

[1] P. Bodik, A. Fox, M. Jordan, D. Patterson, A. Banerjee, R. Jagannathan, T. Su, S. Tenginakai, B. Turner, and B. Ingalls, Advanced Tools for Operators at Amazon.com, *First Workshop on Hot Topics in Autonomic Computing* (HotAC'06), Dublin, Ireland, June 2006

[2] P. Dourish, P., Swinehart, D. and Theimer, M. 2000. The Doctor is In: Helping End-Users Understand the Health of Distributed Systems. *Proc. 11th IFIP/IEEE Workshop on Distributed Systems Operation and Management DSOM 2000* (San Antonio, TX). LNCS 1960, Springer, 157-168.

[3] J. Mackinley, Show Me: Automatic Presentation for Visual Analysis. *Proc. Infovis 2007*.

[4] F. Mansmann, D. Keim, S. North, B. Rexroad, and D. Sheleheda, Visual Analysis of Network Traffic, *Proc. Infovis 2007*.

[5] T. McCartney, K. Goldman, and D. Saff, EUPHORIA: End User Construction of Direct Manipulation User Interfaces for Distributed Algorithms. *Software Concepts and Tools* (16)4:147-159. Springer. 1995.

[6] Y. Moses, Z. Polunsky, A. Tal, and L. Ulitsky, Algorithm Visualization for Distributed Environments. *Proc. Infovis 1998*.

[7] T. Munzner, E. Hoffman, K. Claffy, K., and B. Fenner, Visualizing the global topology of the MBone. *Proc. Infovis 1996*.

[8] G. Robertson, K. Cameron, M. Czerwinski, and D. Robbins, Animated Visualization of Multiple Intersecting Hierarchies, *Information Visualization*, 1, 1, 50-65, Palgrave, April 2002.

[9] J. Thomas and K. Cook. Illuminating the Path. IEEE Computer Society 2005.

[10] K.-P. Yee, D. Fisher, R. Dhamija, and M. Hearst, Animated Exploration of Dynamic Graphs with Radial Layout. *Proc. Infovis 2001*.

[11] Michael Isard, Autopilot: Automatic Data Center Management, *Operating Systems Review* 41(2): 60-67, April 2007.

[12] Patrick Baudisch, Desney Tan, Maxime Collomb, Dan Robbins, Ken Hinckley, Maneesh Agrawala, Shengdong Zhao, Gonzalo Ramos, Phosphor: Explaining Transitions in the User Interface Using Afterglow Effects, *UIST 2006*.

[13] Fox, Armando and D. Patterson, ''Self-Repairing Computers,'' *Scientific American*, June, 2003.Designing & Deploying Internet-Scale Services, James Hamilton, *LISA 2007*, Dallas.