CloudGazer: A Divide-and-Conquer Approach to Monitoring and Optimizing Cloud-Based Networks

Holger Stitz*
Johannes Kepler University Linz

Samuel Gratzl†
Johannes Kepler University Linz

Michael Krieger‡
RISC Software GmbH

Marc Streit§
Johannes Kepler University Linz

Figure 1: *Screenshot of CloudGazer showing the application perspective in the focus view (right) and the virtual and physical perspectives as interactive thumbnails (left). The user has selected the blocks representing 'mail01' and 'wss1_8' to inspect their relationship across the semantic perspectives. By looking at the dynamically created inlay (bottom), it becomes obvious that the virtual machine 'big15' has a high load even though 'wss1_8' has only few connections. The user concludes that another application on 'big15' must cause the problem.*

Abstract

With the rise of virtualization and cloud-based networks of various scales and degrees of complexity, new approaches to managing such infrastructures are required. In these networks, relationships among components can be of arbitrary cardinality (1:1, 1:*n*, *n*:*m*), making it challenging for administrators to investigate which components influence others. In this paper we present *CloudGazer*, a scalable visualization system that allows users to monitor and optimize cloud-based networks effectively to reduce energy consumption and to increase the quality of service. Instead of visualizing the overall network, we split the graph into semantic perspectives that provide a much simpler view of the network. *CloudGazer* is a multiple coordinated view system that visualizes either static or live status information about the components of a perspective while reintroducing lost inter-perspective relationships on demand using dynamically created inlays. We demonstrate the effectiveness of *CloudGazer* in two usage scenarios: The first is based on a real-world network of our domain partners where static performance parameters are used to find an optimal design. In the second scenario we use the VAST 2013 Challenge dataset to demonstrate how the system can be employed with live streaming data.

1 Introduction

The availability of modern cloud computing technology has led to a surge in building more dynamic, fast growing, and continually changing systems. Cloud-based networks are built from various physical components, such as servers and storage devices, that host applications and provide resources that can be used flexibly for different purposes. To make optimal use of the hardware, applications run on virtual machines (VMs) that are, in turn, hosted on servers. However, the assignment between components is neither exclusive nor static. Multiple application instances can run on the same VM, and multiple applications of the same type can run on multiple VMs. Moreover, a physical server can host several VMs. To optimize the quality of service and minimize energy consumption, these assignments are changed regularly depending on the load of individual VMs or other circumstances in the network.

The work of cloud data center administrators comprises many different tasks, ranging from designing the network to active monitoring and optimizing the infrastructure for reduced energy consumption and a high quality of service. State-of-the-art network monitoring systems are often of limited use for these tasks, as they provide only an overview of the status of isolated components, such as CPU load, memory load, and available bandwidth. However, the crucial knowledge about how components influence each other is missing. An alternative approach is to present the overall network infrastructure as a graph. Figure 2 visualizes an example network in which physical components are shown in blue, virtual machines (VMs) in green, and applications in red. As can be seen, the graph can become cluttered quickly—even for small networks.

In cloud-based networks we can differentiate between two basic types of relationship: (1) **direct relationships** between components of the same type, representing physical connections or logical groupings of components (e.g., a grouping of VMs or applications by customer); and (2) **mapping relationships**, representing the assignment of one component to another (a VM running on a server). In Figure 3a and 3b, direct relationships are indicated by solid lines and mapping relationships by dashed lines.

Instead of letting users work with the overall graph that mixes

*e-mail: holger.stitz@jku.at
†e-mail: samuel.gratzl@jku.at
‡e-mail: michael.krieger@risc-software.at
§e-mail: marc.streit@jku.at

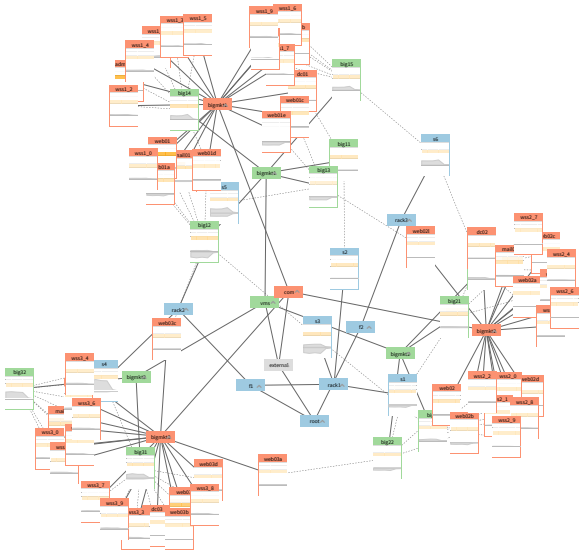
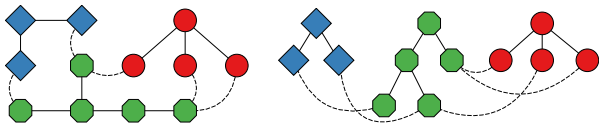


Figure 2: *Graph of a cloud-based network with 67 nodes. Blue, green, and red nodes encode physical components, VMs, and applications respectively. Solid links denote relationships between components of the same type, such as logical groupings of VMs and applications by customer, and dashed links indicate mapping relationships where one type of component is assigned to a component of a different type.*

both relationship types, we split the network into perspectives according to component type: physical, virtual, and application perspective, as demonstrated in Figure 3. The resulting perspectives are much smaller and easier to manage, and also match better the mental model of the administrators. This subdivision strategy for coping with the complexity of graphs has already been applied successfully in many different domains. The large biological pathway network, for instance, is subdivided into small semantic subpathways [12].



(a) *Overall graph* **(b)** *Component-specific perspectives*

Figure 3: *Division of the network into component-specific perspectives. Solid lines represent direct relationships between components, while dashed lines indicate mapping relationships. The graph in (a) is split into the three perspectives shown in (b).*

However, subdividing the network comes at the cost of losing mapping relationships, which are crucial, for instance, to avoiding side effects during optimization that result from changes in the network. For example, migrating a VM to another server can optimize one application's communication, but may hamper the performance of other applications hosted on the same VM.

The primary contribution of this paper is *CloudGazer*, a visualization system for **analyzing**, **monitoring**, and **optimizing** complex distributed systems. *CloudGazer* lets users work with separate perspectives while reintroducing lost inter-perspective relationships on demand. As a secondary contribution we present the *Hierarchical Grid* layout, which further increases the scalability of our solution in terms of the number of components.

2 Domain Background and Goals

Modern networks comprise different types of components that all work together: physical servers, virtual machines (VMs) hosted on servers, and applications running on the VMs. This design results in a graph where relationships among components can be of arbitrary cardinality (1:1, 1:*n*, *n*:*m*). In the following section, we introduce different service models offered by providers, followed by a discussion of the domain goals we aim to solve.

2.1 Cloud Computing Stack

Before cloud computing became popular, customers were able to rent a whole physical server located in some data center. However, with improved virtualization approaches, the rise of cloud computing and platforms such as *VMWare VSphere*[1], *OpenStack*[2], and *OpenNebula*[3] the situation has changed fundamentally. According to the established NIST definition [14], cloud computing can be categorized into three service models: *Infrastructure-as-a-Service (IaaS)*, *Platform-as-a-Service (PaaS)*, and *Software-as-a-Service (SaaS)*.

IaaS providers sell VMs to their customers, who can freely install their preferred operating system, host services, and manage their own software-defined network. Using this strategy, IaaS providers are able to increase their overall data center workload by hosting multiple VMs on a single physical server. This has the advantage that customers are not directly affected by hardware problems. Most of the major cloud operators today (*Amazon Web Services*[4], *Microsoft Azure*[5], *Google Cloud Platform*[6], and *SoftLayer*[7]) provide IaaS for their customers. PaaS providers go one step further and provide only platforms on which customers can run their applications. Examples are classic web-hosting providers and also Microsoft Azure, *Google App Engine*[8], and *IBM BlueMix*[9], which all provide a platform to host websites or web-applications. The last type of cloud provider offers specific applications or software to the customers, which is called SaaS. Customers of such providers do not have any administrative rights and are restricted to using only specific services. A prominent example is the Customer Relationship Management service provided by *Salesforce*[10].

Depending on the cloud computing model, administrators encounter various challenges when managing their networks. While in SaaS scenarios they have full control of every aspect of the network, when renting out servers they can influence only how the underlying physical network is organized. In all other cloud models, administrators can manipulate the assignment of components. In the IaaS case, for instance, they can reassign VMs to servers. The more control customers have, the more they want to monitor, manage, and optimize the network, for example, by moving applications between different rented VMs. However, depending on the assignments between VMs and physical servers, moving an application may decrease the performance of other applications.

In general, customers would benefit from knowing, for instance, the assignments of their VMs to servers. However, for privacy reasons they are often only allowed to see coarse high-level information. Similarly, administrators of IaaS or PaaS providers would benefit from knowing details about applications run by their customers, to optimize the assignment for quality of service and energy consumption.

[1] http://www.vmware.com/products/vsphere/
[2] http://www.openstack.org/
[3] http://www.opennebula.org/
[4] http://aws.amazon.com/
[5] http://azure.microsoft.com/
[6] http://cloud.google.com/
[7] http://www.softlayer.com/
[8] http://appengine.google.com/
[9] http://ace.ng.bluemix.net/
[10] http://www.salesforce.com/

2.2 Goals

Over several months of close cooperation with our project partners, we analyzed the process of managing and optimizing cloud-based networks. There are commercial products in this field such as *VMWare's Distributed Resource Scheduler*[11], which try to optimize assignments of VMs to servers by analyzing their behavior automatically. However, such systems are of limited use for complex, heterogeneous cloud-based networks. They apply somewhat simplistic models and rules for optimization and work best in cases where all VMs are clones, as in a group of web servers. In heterogeneous networks, a deep understanding of the semantics and communication between the components from all three perspectives (physical, virtual, and application) is important to monitor and optimize the network effectively. Consequently, a visualization solution that targets these problems should allow administrators to:

G I: Monitor the status of the network by visually inspecting static performance information about the components (e.g., CPU power, available memory) and/or live performance and traffic data.

G II: Discover bottlenecks by analyzing the infrastructure's design in the context of the monitoring information.

G III: Optimize the network interactively. Depending on the requirements and purpose of the network, different optimization criteria exist. For example, if the internal communication needs to be minimized, the administrator's goal is to reduce the length of communication routes between components. If the task is to optimize the balance of resources, administrators should be able to change the mapping between components, e.g., the assignment of VMs to physical servers or assignment of applications to VMs.

3 Requirements

Below we present a list of requirements that an effective cloud monitoring and optimization solution must fulfill. We elicited the requirements in interviews and feedback sessions with cloud computing experts, one of whom is co-author of this paper.

R I: Encode topology of cloud infrastructure. The visual representation of the cloud-based network needs to show relationships between components and encode different types of components.

R II: Encode static or dynamic attributes. This includes static performance attributes such as installed main memory, hard disk capacity, and CPU specification. In the case of dynamically changing data, the visualization must encode attributes such as the current CPU load or main memory load factor. In addition to attributes of single components, the communication flow and connections between components needs to be represented effectively without cluttering the visualization.

R III: Enable time navigation. The user must be able to select interactively the time interval for which the streaming data is encoded in the network visualization. The selected time span should be either bound to the current time point or fixed to a static snapshot of the network.

R IV: Support interactive changes of mapping relationships. It should be possible for users to optimize the cloud-based network by manipulating the mapping relationships between components.

R V: Scalability. The visualization needs to scale to a large number of components, many attributes, and a high traffic load.

R VI: Encode topological evolution. An effective solution should enable users to explore, compare, and analyze changes within the structure and assignments in the network over time.

R VII: Support privacy preservation. Administrators who are in charge of specific sub-parts of the network may not have the clearance to see all parts, but must be offered a privacy-preserving view, in order to minimize side effects when optimizing their part of the network.

[11] http://www.vmware.com/products/vsphere/

4 Related Work

CloudGazer is designed to address the three domain-specific goals formulated in Section 2.2. In this section we start with a discussion of commercial tools that target similar goals, followed by a consideration of related work in network traffic visualization. Finally, we summarize contextually relevant approaches to visually comparing and relating multiple hierarchies.

4.1 Cloud Computing Software

The majority of commercial tools follow a classic dashboard approach that enables users to monitor the current state of cloud-based networks (cf. goal G I). Dashboards are mash-ups of simple graphs, statistical plots, and tables that present the network topology together with traffic and performance parameters over time. In most tools the dashboard is designed to provide a high-level overview of the network, from which users can drill down to lower-level information, such as single transaction events. Examples of such monitoring tools are *OPSView Server Virtualization Monitoring*[12] and *Compuware APM for Enterprise Tiers*[13]. Depending on the tool, information is presented at various levels of detail. The most condensed status of a network or components within the network are traffic-light representations. An inherent problem in many tools is that switching to more detailed information about one component or part of the network often results in a loss of context.

In general, dashboard solutions—if well designed—are well suited to addressing monitoring tasks (goal G I). However, discovering potential bottlenecks (goal G II) is difficult, since individual dashboard elements are often isolated from each other, which hampers the detection of relationships and anti-patterns that could cause problems in the near future. Most of the tools do not focus on integrated ways of optimizing and fixing problems (G III). One exception is *Cirba Control Console*[14], which gives hints on how to optimize the cloud-based network in order to prevent future problems and to increase cost-effectiveness. The hints are based on scores that are computed for all physical and virtual machines. Although the tool supports the task of optimizing the network based on static data effectively, it cannot cope with live streaming data.

Tools such as OPSView Server Virtualization Monitoring and Compuware APM for Enterprise Tiers present the overall structure of the cloud-based network in a single graph or tree representation. Although this works for small networks, it results in scalability issues with a growing number of components. Larger graphs get cluttered easily, making it hard for administrators to interpret and relate different components and their relationships in the context of live streaming data.

In summary, none of the available tools addresses all three goals effectively in a single solution. We therefore believe that the presented solution could have a significant impact on the design of next-generation cloud computing tools.

4.2 Network Traffic Visualization

The problem of visualizing computer networks has been and remains an active research topic in the visualization community. Most of this work focuses on the task of monitoring and analyzing traffic visually, for instance, to detect and react to attacks. Examples are the work by Fisher et. al that shows connections on top of a treemap which encodes the subnets of the network [6], and the *LiveRAC* system [13], which uses a space-filling layout for visualizing the status of nodes at multiple levels of detail over time. LiveRAC is particularly interesting, as it scales well to visualizing data associated with thousands of nodes. In LiveRAC and many other systems, the topology is secondary and often not even shown in the visualization.

[12] http://www.opsview.com/virtualization-monitoring/

[13] http://compuware.com/

[14] http://www.cirba.com/

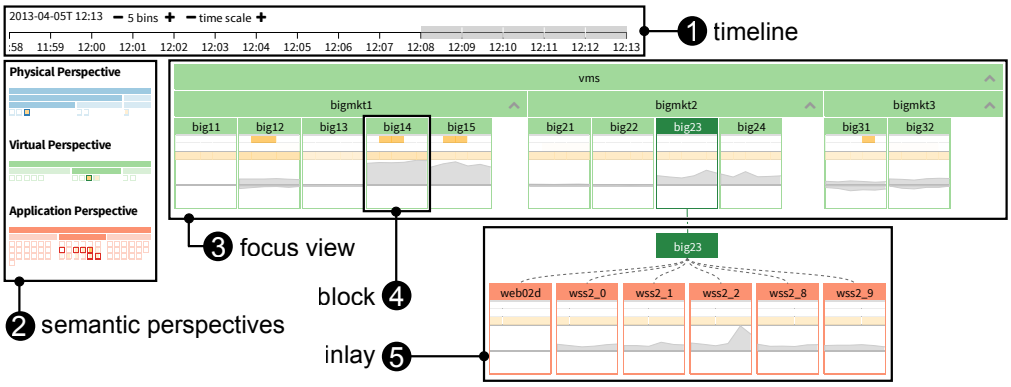


Figure 4: *Building blocks of CloudGazer. (1) Timeline for temporal navigation. (2) Semantic perspectives shown as interactive thumbnails. (3) Focus view presenting one perspective in greater detail. (4) Blocks visualizing a single component with its associated data. (5) Inlay showing relationships of selected nodes to other perspectives.*

However, from a domain-independent visualization point of view, it boils down to the challenge of presenting topological information of a graph or hierarchy together with node and edge attributes that potentially change over time. Existing solutions usually focus either on the topology aspect (e.g., [16]) or on the evolution of nodes and attributes over time (e.g., [13]). A notable exception is *enRoute* [17], where users are able to select a path in a biological network, which is then presented together with associated experimental data. Another exception is the work by Saraiya et al. [19] that shows heatmaps and line charts as small glyph nodes embedded in a graph visualization. However, the solution becomes cluttered quickly if applied to graphs with more than a few dozen nodes. In *CloudGazer*, we strive to incorporate both aspects—topology and additional data attributes—while addressing the scalability issue by splitting the network into multiple perspectives.

4.3 Hierarchy Matching

As cloud-based networks are graphs, the vast body of work on graph visualization is applicable [22]. Due to the fact that the workflows and the associated data change over time, also the state-of-the-art in the sub-field of dynamic graph visualization is relevant in this context [1, 10]. However, instead of visualizing the overall graph, we cope with the complexity of the cloud-based network by subdividing it into multiple hierarchies. This strategy requires reintroducing the lost relationships between the hierarchies. A vast body of related work exists on matching and comparing two or multiple hierarchies. A recent survey by Graham et al. [7] identified seven fundamental approaches to this task: i) drawing edges between spatially separate hierarchies; ii) highlighting related nodes; iii) animating between the hierarchy representations; iv) using matrix representations; v) agglomerating nodes that have multiple parents for display in the same representation; vi) 3D representation of interlinked hierarchies; vii) atomic view that shows only parts of the hierarchies on demand.

The first two approaches are options that we discuss in further detail in the remainder of this section. All other approaches are not applicable. Animation (iii) and atomic views (vii) are not viable options, as administrators need to see the status of all perspectives concurrently to be able to monitor them (see G I). Matrix representations (iv), such as the *RelEx* system [20], can match only two trees and are therefore not applicable in this context. Agglomerating nodes (v) can also be ruled out, as it would increase the complexity of the perspectives again. 3D representations (vi) suffer from occlusion and perspective distortion.

The first approach of drawing edges between hierarchies is well suited to identifying structural changes. *TreeJuxtaposer* [15], for instance, supports pairwise tree comparison. The work by Robertson et al. [18] follows a similar idea for mapping two schema trees. Holten and Wijk [9] visualize two trees as space-filling icicle plots that face each other, where items between plots are connected by hierarchically bundled edges. Another example is *Code Flow* [21], which visualizes drifts, merges, and insertions between different versions of source code. The conceptual difference from the previous examples is that Code Flow visualizes the evolution between multiple states of the tree by showing each state in a parallel coordinate fashion. Although this extends the approach to multiple trees, it has the same limitations as parallel coordinates: only relationships between adjacent trees can be seen.

All these papers are good examples of how to address the comparison task effectively. In *CloudGazer*, however, we not only have direct 1:1 relationships between trees but inter-hierarchy relationships of varying cardinality (1:*n* and *n*:*m*), which makes it hard for users to understand the complex relationships between the trees. Thus, we use dynamically created inlays to address this problem (see Section 5.4).

The second approach identified in the survey uses coloring and highlighting to visualize relationships across trees. Bremm et al. [3] proposed an interactive visual comparison of multiple trees where users need to select one tree as reference in order to see how it relates to others. All compared trees are presented as small views rendered next to each other. While this allows users to identify topological differences between trees, it is error-prone and slow, as it requires users to manually match the relationships by visually comparing them—a task that is known to be cognitively demanding. In *CloudGazer* we utilize interactive thumbnails to provide an overview of the perspectives. However, for communicating inter-perspective relationships, we rely on inlays that contain all relationships relevant to the current selection.

5 CloudGazer Visualization Approach

Even small-scale networks with a few dozen components can become hard to understand, as demonstrated in Figure 2. To address this issue, we apply a divide-and-conquer strategy where the overall graph of the network is broken up into component-specific perspectives (see Figure 3). Deriving perspectives from the overall graph is a one-time authoring step that can be performed automatically.

In *CloudGazer* we arrange the perspectives in a multiple coordinated view setup with each perspective shown as a separate view. The user can interactively choose a focus perspective that is presented in detail, while other perspectives are shown as interactive thumbnails. As illustrated in Figure 4, *CloudGazer* consists of *blocks* encoding component-specific attributes (see Section 5.1), *interactive thumbnails* showing a high-level version of all perspectives, a *focus view* visualizing one perspective in detail (Section 5.2), *inlays* embedded in the focus view showing relationships of the focus perspective to others (Section 5.4), and a *timeline* for temporal navigation (Section 5.5).

By selecting components of interest, the user can investigate mapping relationships across perspectives effectively. We insert these relationships into foreign perspectives as inlays. To make the association between nodes and perspectives clear, we assign the same color to all nodes that belong to a particular perspective. In the following sections we discuss the building blocks of the *CloudGazer* system.

5.1 Blocks

Blocks are the basic visual unit of *CloudGazer* that facilitate monitoring the state of a single component, as illustrated in Figure 5. Depending on the usage scenario, a block encodes static performance information of components or live status and traffic information (see R II).

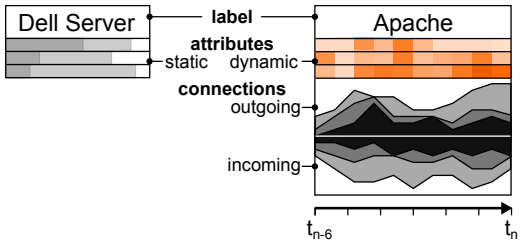


Figure 5: *Blocks represent the status of a single component. Static blocks (left) encode component attributes using stacked bars. Each bar corresponds to one attribute. Dynamic blocks (right) visualize live streaming data using heatmaps and streamgraphs. Each row of the heatmap encodes data from different attributes over time. In the case of live streaming data, new data is pushed into the heatmap from the right. The streamgraph in the lower part of the representation encodes incoming and outgoing connections. The height of the first inner layer corresponds to the number of connections with directly linked components. With each layer the distance in the hierarchy increases, as indicated by a decreasing brightness.*

Stacked Bars We represent static component attributes as stacked bars normalized by their global maximum value (see Figure 5). The light gray bar encodes a value associated with the component, while the dark bar represents the sum of attribute values from other perspectives that are assigned to this component. For example, if the mapped attribute is memory, the length of the bar encodes the main memory installed on the server. While the dark gray bar is the memory allocated by all VMs hosted on this server, the light gray bar encodes the remaining free memory. To make the length of the bars comparable across components, we use the white portion of the bar to indicate a difference to the maximum value across all components on this perspective level.

Heatmap We use heatmaps to encode component attributes that change over time (see Figure 5). Each column represents a time step, and each row is associated with a different attribute. Possible attributes are CPU load, main memory load factor, and hard drive disk usage. New live traffic data is added as the last column on the right, pushing previous time steps one column to the left. The number of columns can be changed interactively by the user. We use a white (0) to orange (1) color scale.

Streamgraph We use streamgraphs to encode the communication with other components, as illustrated in Figure 5. Instead of explicitly visualizing the communication between individual components by changing the edge encoding, we group connections according to the number of intermediate hops in the perspective hierarchy and show the groups as layers of the streamgraph. Parent and child components, for instance, have a distance of one, while siblings and grandparents/grandchildren have a distance of two. All external communication with components outside the cloud-based network is summarized as the outermost layer. All groups are stacked according to their distance and normalized by their current global maximum value. In addition, we differentiate between incoming and outgoing connections by separating them into two charts, as show in Figure 4. Streams pointing upwards and downwards represent outgoing and incoming connections, respectively. Note that, depending on the user's preferences, it is possible to switch from streamgraphs to a stacked bar chart representation. Due to our design decision to encode communication in the block rather than the edges, point-to-point connections are lost. To alleviate this problem, a user can select a block in order to filter the data of all other blocks to contain only the communication with that selected.

As the streamgraph is the most salient part of the block representation, it should encode the attribute that is most relevant to solving the analysis task. In our usage scenarios, streamgraphs represent the number of connections, while the evolution of all other attributes is visualized in the heatmap. However, the mapping of attributes to the streamgraph and rows of the heatmap can be tailored to the usage scenario. Multiple stacked streamgraphs are also possible.

Note that we use the color of blocks contained in the interactive thumbnail perspectives to encode a single attribute. In the examples shown in the paper, the color represents an aggregated value of the number of connections.

5.2 Focus View

The focus view is the central visualization in *CloudGazer* and presents in detail the currently selected perspective, as shown in Figure 4. The visualization of the focus perspective is linked with the thumbnails of all perspectives shown on the left of the interface. When the user selects a block in the focus view, all blocks in foreign perspectives that share a mapping relationship are highlighted in the interactive thumbnails, as shown in Figure 1.

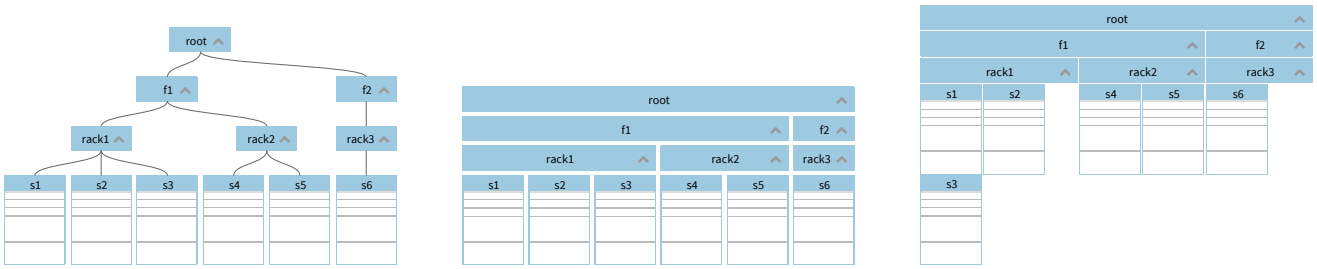
CloudGazer supports various layouts for arranging the blocks effectively (addressing requirement R I). Widely used tree layouts such as node-link and icicle plots [11] have the disadvantage that they grow in size rapidly with an increasing number of leaf nodes. To alleviate the problem, we propose the *Hierarchical Grid* layout, which is explained in more detail in the following section. To account for the size of the perspective and the task at hand, users can switch freely between layouts.

Even with a space-efficient layout and dividing the network into multiple smaller perspectives, the most crucial issue of the focus view is its scalability to larger numbers of blocks without sacrificing the ability to track and monitor individual components (see also requirement R V). In *CloudGazer* we take a series of measures to address this issue:

Collapse & Expand By clicking on nodes, users can collapse all child nodes to a single node. For example, in the application perspective all web applications can be collapsed on demand. The collapsed proxy node then shows aggregated information from all hidden child nodes.

Hierarchical Zooming To quickly focus on certain sub-parts of the hierarchy, users can double-click on a node to turn into the new root of the displayed hierarchy. This kind of navigation is particularly useful for large hierarchies.

Activity-Based Shrinking In real-world scenarios not every component will be active. Depending on the current load of the network, some components might have little or no communication at all. These components are less relevant to administrators. Thus, they can be visualized in a simplified and more compact form. For these cases, older time steps can be removed, to make the blocks thinner, or blocks can even be hidden. *CloudGazer* optionally provides automatic adaptation of component width according to its current activity, calculated by the attributes' variance over time. The thinner a block, the less of its attribute's history is shown, which ensures that blocks remain comparable. In addition, the blocks can be



(a) *Node-link layout* **(b)** *Icicle plot layout* **(c)** *Hierarchical Grid layout*

Figure 6: *Hierarchy represented by the different layout approaches available in CloudGazer.*

automatically ordered by activity, such that active ones are moved to the front, the downside of which is a constantly changing layout.

5.3 Perspective Layouts

An effective layout for arranging hierarchically structured perspectives is an important success factor for a cloud-based network visualization. For smaller hierarchies we provide a regular node-link tree layout (see Figure 6a). To increase the scalability in terms of leaf nodes (R V), we introduce the *Hierarchical Grid* layout.

The Hierarchical Grid layout is a modified version of an icicle plot [11]. As icicle plots are space filling, the node widths on each level of the hierarchy are determined by dividing the available width by the number of nodes. Figure 6b shows a small hierarchy represented as an icicle plot. However, with an increasing number of nodes, the node width can become too small. As we visualize traffic data and additional attributes inside the nodes, a reduced node width also reduces the space for visualizing data. In classic icicle plot implementations, users can alleviate the problem by zooming into a part of the hierarchy by promoting a node to become the new root. In the Hierarchical Grid layout we arrange the leaf nodes in a grid-like structure. While this causes the representation to grow downwards, the node width for the leaves is kept constant, as can be seen in Figure 6c. A constant node width is important to make the data shown in the streamgraphs and heatmaps comparable across blocks. Figure 1 shows the layout applied to an application perspective with 54 blocks. The design decision to keep the node width static in a space-filling layout can result in empty space between branches of the hierarchy. However, we consider this to be a minor esthetic issue that does not have a negative impact on functionality.

Both icicle plots and the Hierarchical Grid layout express the hierarchy implicitly through the position of the nodes. If nodes from different levels touch each other, they share a relationship. As traffic information is encoded in the nodes themselves and not on the edges, icicle plots and the Hierarchical Grid layout are more space efficient than explicit node-link diagrams. Due to the compactness of the Hierarchical Grid and the fact that it can also be interpreted when rendered smaller, we use this layout for the interactive thumbnails of the perspectives, as shown on the left in Figure 1.

Note that in this paper we focus on hierarchically structured perspectives. However, the presented visualization concept is independent of structure and layout of the perspectives and could also be applied to general graphs or other specialized topologies.

5.4 Inlays

A downside of splitting the overall network into multiple perspectives is the loss of visual representation of the mapping relationships between components belonging to different perspectives. *CloudGazer* addresses this problem by letting the administrator select blocks for which the lost context will be reintroduced using *inlays*. Inlays are dynamically created graphs that are assembled according to the current block selection. When the user selects a single component in one perspective, all related components from other perspectives are added to the inlay graph. If the selected block is a server, for instance, the inlay contains all VMs hosted by this server and all applications running on the VMs (see Figure 7). If the user selects a second block, the inlay algorithm looks for a path that connects the selected components in the other perspectives (R I). If a path can be found, all components along the path will be part of the inlay, as can be seen in Figure 1.

In the focus view, which shows the selected perspective, we add the inlay at the bottom. If the inlay shows the path between two selected blocks across multiple perspectives, the left-most and right-most blocks are duplicates of the originally selected block. We use dotted lines to connect the original blocks with the duplicates in the inlay. Animated transitions [8] help users to track visual state changes when adding the inlays.

By inspecting components and relationships in inlays, administrators can discover bottlenecks in the cloud infrastructure (addressing G II). In addition, *CloudGazer* enables users to optimize the network proactively by changing mapping relationships (G III). Using drag-and-drop makes it possible to reassign components across perspectives (fulfilling R IV). If blocks visualize live traffic data, the impact of the changes can be observed immediately.

5.5 Timeline

The interactive timeline provides for temporal navigation and for choosing the time span that shows up in blocks (R III). The length of the time span directly influences the block width. The time span is discretized into multiple bins, as indicated within the selected time span shown in the top of Figure 4. The number of discrete steps can be changed interactively and is used within blocks to bin attribute values.

5.6 Implementation

CloudGazer is an HTML5 web application that uses D3 [2] for visualization and the AngularJS[15] web framework to mash up elements. The server part is written in Python using the Tornado framework[16] to provide live traffic data. The interaction with the prototype system is demonstrated in an accompanying video.

6 Usage Scenarios

We demonstrate the effectiveness of *CloudGazer* in two scenarios. The first discusses how the prototype system can be used to optimize statically a cloud-based network of one of our partners by optimizing assignments between components (G III). The second uses simulated data to demonstrate how *CloudGazer* can be used to monitor dynamic networks (G I) and discover bottlenecks (G II).

6.1 Optimizing a Cloud-Based Network

Our project partner *RISC Software GmbH* specializes in administrating various cloud infrastructures for different customers and research projects. They maintain an IBM CloudBurst with four physical servers. Each CloudBurst server has 72 GB memory, is

[15] http://angularjs.org/
[16] http://tornadoweb.org/

(a) *Before optimization* **(b)** *After optimization*

Figure 7: *Server perspective with the selected server 'cb2' and the corresponding inlay with related VMs and applications. The two stacked bars encode the components' main memory and disk space (dark gray = used, light gray = free, white = empty space to make bars comparable across components, i.e., only present if available memory is different between components on the same hierarchy level). To reduce the connection distances between the VMs 'tele_1', 'tele_2', and 'tele_3', the administrator reassigns the VM 'tele_2' to server 'rsw1' via drag-and-drop.*

connected to a 40 TB storage array, and has a 10 GBit connection. The VMs have 512 MB to 32 GB of memory assigned, which can be flexibly allocated. In addition, they maintain a second rack with four servers, each with a configuration of 128 GB memory, 2 x 1 TB internal storage, and a 10 GBit connection.

A research project on traffic engineering is collecting and processing telematics data. Three applications are required for this purpose: a database, a computation server, and a web server, which initially run on independent VMs (*tele_1*, *tele_2*, *tele_3*) and on separate CloudBurst servers (*cb1*, *cb2*, *cb3*). However, the data transfer between the database and the computation server results in high internal network load. The administrator decides to merge the three applications on one physical server in order to reduce communication distances. Figure 7 shows the server perspective in focus with related VMs and applications running on the selected server *cb2* as inlay. The visualization in *CloudGazer* shows that the CloudBurst servers have insufficient memory capacity to merge all VMs on one single server. The administrator explores the servers of the other rack and discovers that server *rsw1* has enough available memory for hosting all project-related applications. Using drag-and-drop, he assigns the VM of each CloudBurst server to server *rsw1*.

6.2 Monitoring Dynamic Cloud-Based Network

In the second usage scenario we demonstrate the *CloudGazer* system with simulated data from the 2013 Big Marketing VAST Mini-Challenge [4]. The dataset consists of NetFlow data along with additional server attributes, including CPU load and memory usage, collected over a period of two weeks. We interpret the given network infrastructure as the application perspective. Based on this dataset, we generated a virtual and physical perspective. Since the data are relatively sparse, we aggregated them such that one second in the visualization corresponds to 60 seconds in the dataset. Further, we combined all workstations of each Big Market section in ten characteristic workstations running on terminal servers. Together with cloud computing experts we created the following use case to demonstrate how *CloudGazer* supports administrators monitoring a network based on live data:

The administrator of the Big Market network is responsible for handling customer requests concerning problems with the cloud infrastructure. A customer reports a problem accessing her e-mail and other applications. The administrator starts to investigate the issue by looking at internal logs. He finds out that the customer is logged in as *wss1_8* and decides to look at the status of the application in *CloudGazer*'s application perspective (see Figure 1). Each block in the focus view shows the overall status of an application, including CPU load and disk usage as heatmaps and incoming/outgoing connections as streamgraphs. However, neither *wss1_8* nor the mail server *mail01* are under heavy load according to the block visualizations. There is some external traffic on *wss1_8*, but this seems to be regular traffic caused by the customer's web usage. The administrator suspects that not the applications themselves are the problem but the VMs they are running on, and in particular their physical relationship. By selecting both blocks *mail01* and *wss1_8* in the focus view, an inlay is added showing with which VMs and servers the applications are associated. He realizes that *big15*, which hosts the customer's workstation, is under heavy load, as shown in Figure 1. The administrator clicks on the VM block, which makes *CloudGazer* switch to the virtual perspective that shows an inlay of all applications hosted on the selected VM. The administrator realizes that one application, *wss1_6*, is consuming most of the VM's resources. Using drag-and-drop, he moves this application to an idle VM. This solves the customer's problem, since the VM can now provide more resources to the workstation, and the overall network is again more balanced.

7 Discussion and Limitations

Scalability Scalability is the most critical concern when developing monitoring visualization techniques for large-scale cloud infrastructures. The strategy of splitting the network into semantic perspectives alleviates the problem but does not conclusively solve it. To further increase the scalability of *CloudGazer* so it can cope with large cloud-based networks, we take several measures, the most important of which is an optimized layout (Section 5.3) and specially designed interaction techniques (Section 5.2).

While the strategy of breaking up a cloud-based network into smaller semantic perspectives increases its scalability to larger networks, it also introduces problems concerning loss of relationship representations. In *CloudGazer* we address this issue by adding inlays that show relevant portions of related perspectives on demand.

Communication Encoding Communication between components is mainly 1:1. A naive approach is to visualize all connections using a node-link diagram and encode the amount of traffic by changing the width or color of edges. However, in discussions with our project partners we found that the communication distance, i.e., the number of intermediate hops, is more relevant than the actual communication endpoint. Therefore, we use streamgraphs to encode communication distances for each component (see Section 5.1). When a user selects a component, we filter all other streamgraphs to show only the communication with that component, allowing users to inspect the 1:1 connections on demand.

Optimization Costs *CloudGazer* allows administrators to optimize their networks interactively by manipulating the assignments

of components across different perspectives. For example, administrators can move one VM to another server in order to optimize the communication distances or average server load. However, moving a VM to another server entails costs, such as the transfer time from one server to another or a possible short outage of the VM. In the current version of *CloudGazer* these costs are not considered, even though they can influence the usefulness of optimizations in terms of cost/benefit ratio.

Evolution Monitoring a network consists both of tracking the status of its components and of monitoring for changes in its topology due to reconfiguration in response to changed requirements or to optimization measures. Thus, perspectives and the mappings between them also change over time. Finding ways to let administrators track changes and evaluate their consequences is an interesting research question (R VI), which we plan to address in the future.

Privacy Preservation In large-scale networks, multiple administrators work on different parts or aspects of the same network. Some users may have limited clearance to see certain parts of the network. However, to avoid side effects during concurrent optimizations, it is beneficial to give them an overview of the whole network. Privacy-preserving visualization techniques need to be applied to provide abstract overviews without showing details that are not allowed to be seen by certain users. A simple approach is to hide labels. However, even without labels individual components might be identifiable due to characteristic communication or attribute patterns. Consequently, more sophisticated privacy-preserving visualization techniques must be integrated [5]. *CloudGazer* does not yet include such measures. Therefore, requirement R VI remains open for future work. In particular, the problem of finding the right balance between costs and benefits of privacy-preserving network visualizations is an interesting topic.

8 Conclusions and Future Work

We have presented *CloudGazer*, a flexible visualization solution for monitoring and optimizing cloud-based networks. Following the divide-and-conquer principle, we first divide the overall network into smaller semantic perspectives that are easier to understand and handle. In a second step, we reintroduce the lost inter-perspective relationships for selected parts of the focus perspective by adding dynamic inlays.

As part of future work we plan to increase the flexibility of our approach by generalizing the topology of perspectives from hierarchies to general graphs. This is particularly relevant for covering large-scale infrastructures where many redundant components and connections exist in order to increase reliability and system stability. Another interesting open research direction is tracking and visualizing the evolution of networks in terms of attributes (traffic) and topological changes, including altered relationships between perspectives. Applying methods from predictive analytics to this provenance information would enable administrators to pinpoint potential future bottlenecks. In a next step, the analytical processing results could also be used to make suggestions, such as how and when the topology or the mapping between components should be optimized, and what the implications of the changes would be. Ultimately, this would enable administrators not only to react to current problems but to prevent them proactively.

Acknowledgements

This work was funded by the Austrian Research Promotion Agency (840232).

References

[1] F. Beck, M. Burch, S. Diehl, and D. Weiskopf. The state of the art in visualizing dynamic graphs. In *Proceedings of the Eurographics Conference on Visualization (EuroVis '14) – State of The Art Reports*, 2014.

[2] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '11)*, 17(12):2301–2309, 2011.

[3] S. Bremm, T. von Landesberger, M. Hess, T. Schreck, P. Weil, and K. Hamacherk. Interactive visual comparison of multiple trees. In *Proceedings of the IEEE Conference on Visual Analytics Science and Technology (VAST '11)*, pages 31–40. IEEE, 2011.

[4] K. Cook, G. Grinstein, and M. Whiting. VAST challenge dataset 2013, mini-challenge 3, 2013.

[5] A. Dasgupta and R. Kosara. Adaptive privacy-preserving visualization using parallel coordinates. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '11)*, 17(12):2241–2248, 2011.

[6] F. Fischer, F. Mansmann, D. A. Keim, S. Pietzko, and M. Waldvogel. Large-scale network monitoring for visual analysis of attacks. In *Proceedings of the Workshop on Visualization for Computer Security (VizSec '08)*, pages 111–118. Springer, 2008.

[7] M. Graham and J. Kennedy. A survey of multiple tree visualisation. *Information Visualization*, 9(4):235–252, 2010.

[8] J. Heer and G. G. Robertson. Animated transitions in statistical data graphics. *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '07)*, 13(6):1240–1247, 2007.

[9] D. Holten and J. J. v. Wijk. Visual comparison of hierarchically organized data. *Computer Graphics Forum (EuroVis '08)*, 27(3):759–766, 2008.

[10] N. Kerracher, J. Kennedy, and K. Chalmers. The design space of temporal graph visualisation. In *Proceedings of the Eurographics Conference on Visualization (EuroVis '14, Short Papers Track)*, 2014.

[11] J. B. Kruskal and J. M. Landwehr. Icicle plots: Better displays for hierarchical clustering. *The American Statistician*, 37(2):162, 1983.

[12] A. Lex, C. Partl, D. Kalkofen, M. Streit, S. Gratzl, A. M. Wasserman, D. Schmalstieg, and H. Pfister. Entourage: Visualizing relationships between biological pathways using contextual subsets. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '13)*, 19(12):2536–2545, 2013.

[13] P. McLachlan, T. Munzner, E. Koutsofios, and S. North. LiveRAC: Interactive visual exploration of system management time-series data. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*, pages 1483–1492. ACM, 2008.

[14] P. Mell and T. Grance. The NIST definition of cloud computing, 2011.

[15] T. Munzner, F. Guimbretière, S. Tasiran, L. Zhang, and Y. Zhou. TreeJuxtaposer: Scalable tree comparison using focus+context with guaranteed visibility. In *Proceedings of the ACM Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '03)*, pages 453–462. ACM, 2003.

[16] G. Namata, B. Staats, and B. Shneiderman. DualNet: A coordinated view approach to network visualization. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM '07)*. ACM, 2007.

[17] C. Partl, A. Lex, M. Streit, D. Kalkofen, K. Kashofer, and D. Schmalstieg. enRoute: Dynamic path extraction from biological pathway maps for exploring heterogeneous experimental datasets. *BMC Bioinformatics*, 14(Suppl 19):S3, 2013.

[18] G. G. Robertson, M. P. Czerwinski, and J. E. Churchill. Visualization of mappings between schemas. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI '05)*, pages 431–439. ACM, 2005.

[19] P. Saraiya, P. Lee, and C. North. Visualization of graphs with associated timeseries data. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '05)*, pages 225–232. IEEE, 2005.

[20] M. Sedlmair, A. Frank, T. Munzner, and A. Butz. Relex: Visualization for actively changing overlay network specifications. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '12)*, 18(12):2729–2738, 2012.

[21] A. Telea and D. Auber. Code flows: Visualizing structural evolution of source code. *Computer Graphics Forum (EuroVis '08)*, 27(3):831–838, 2008.

[22] T. von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. van Wijk, J.-D. Fekete, and D. Fellner. Visual analysis of large graphs: State-of-the-art and future research challenges. *Computer Graphics Forum*, 30(6):1719–1749, 2011.