bwVisu: A Scalable Remote Visualization Service and its Application to Flow Visualization

Aksel Alpay[1], Karsten Hanser[2], Egzon Miftari[1], Dennis Schridde[1], Sabine Richling[1], Martin Baumann[1], Filip Sadlo[2] and Vincent Heuveline[1]

[1]Heidelberg University Computing Centre

[2]Interdisciplinary Center for Scientific Computing, Heidelberg University

bwVisu combines an intuitive web frontend with well-established HPC technologies providing an easy-to-use and powerful remote visualization solution. We present bwVisu at the example of the finite-time Lyapunov exponent, a contemporary visualization method for the analysis of time-dependent flow.

1. Introduction

As computational power grows, so does the size and complexity of scientific data. In addition, the transfer of large data is cumbersome, and usually requires centralized services for the secure storage of scientific data. The visualization of this data is therefore increasingly difficult and can hardly be handled by local workstations with their limited computing power. Remote visualization, i.e., bringing the visualization techniques to the data and transferring only the derived visual representations to the user, is therefore steadily gaining importance.

Remote visualization services such as bwVisu stand in between data acquisition and data archiving in the data life cycle: They allow for a convenient way of analyzing and exploring the data after it has been created and is still located on hot storage.

bwVisu aims at providing scientists from Baden-Württemberg with a scalable service for the remote visualization of scientific data, while focusing on the user experience and ease of use. Section 2 describes the underlying hardware of bwVisu. Section 3 details the middleware that forms the interface between the bwVisu hardware and bwVisu Web. In Section 4, we go into more detail on bwVisu Web and its functionality, while Section 5 exemplifies bwVisu at the example of time-dependent flow visualization.

2. Cluster

For bwVisu, a dedicated visualization cluster is available. The cluster consists of eight compute nodes and two login nodes. Each compute node houses 28 CPU cores and two specialized GPUs, which are particularly suited to serving many concurrent users by means of dedicated virtualization functions. The nodes are connected with an Infiniband

Aksel Alpay et al.: bwVisu: A Scalable Remote Visualization Service and its Application to Flow Visualization.
In: Heuveline, V., Gebhart, F. und Mohammadianbisheh, N. (Hrsg.): E-Science-Tage 2019 – Data to Knowledge,
Heidelberg: heiBOOKS, 2020, 173–184. DOI: https://doi.org/10.11588/heibooks.598.c8426

network, and two high-bandwidth uplinks are available for the login nodes. This guarantees fast access for jobs running on the bwVisu visualization cluster to data on other nearby systems, such as the bwForCluster MLS+WISO [19] or SDS@hd [3]. The system is running CentOS and uses the widespread HPC job scheduler Slurm [14].

3. Middleware

A custom middleware provides an interface for the web frontend to manage the visualization jobs. The middleware, called bwVisu *runner*, is written in Python and exposes its features via a flask-based [18] HTTP REST API to the web frontend.

The middleware is designed to be minimally invasive and can be deployed either on nodes of the cluster or on an external system. Figure 1 illustrates the architecture of the bwVisu stack. For job management, the middleware communicates via `ssh` with a set of *command nodes*, where it starts, stops, and monitors jobs with the Slurm commands `sbatch`, `squeue`, and `scancel`. Additionally, the middleware uses `ssh` to access a set of *gateway nodes*, where it establishes forwarding rules of network traffic from ports to the individual compute nodes where the user application runs. This allows the user to connect to his job simply by connecting to a specific network port on one of the gateway nodes. Authentication is handled by the application of the user (e.g., Xpra). In order to scale to many concurrent jobs, the middleware automatically distributes the network load of the jobs across the available gateway nodes.

The state of jobs is continuously monitored from the gateway nodes. This allows the middleware to provide information to the web frontend whether the job is ready to interact with the user, or whether the user has to wait before the job can be accessed (e.g., because it is queued and not yet running, or because it is running but still in an initial start-up phase where network connections are not yet accepted).

All state of the middleware is stored in an `etcd` database, which is a distributed key-value store. Because of the distributed design of `etcd`, it is easy to operate in a redundant manner, and hence allows for a resilient, highly-available deployment of the middleware.

The requirements that the bwVisu middleware imposes on the cluster are designed to be as low as possible: The middleware must be granted ssh access to a dedicated user account, and this account must be able to run Slurm commands on the command nodes and iptables on the gateway nodes[1]. Since the installation of the bwVisu middleware is the only requirement to use a given HPC cluster for bwVisu, extending an existing HPC cluster with bwVisu's remote visualization capabilities is easy and does not necessitate any changes to the existing cluster configuration.

4. bwVisu Web

bwVisu includes the web user interface "bwVisu Web", which allows researchers to start their individual visualization software easily and connect to it using nothing more than

[1] Note that the gateway nodes can also be virtual machines if it is not desired that iptables rules of physical machines in the cluster are modified by the bwVisu software stack.

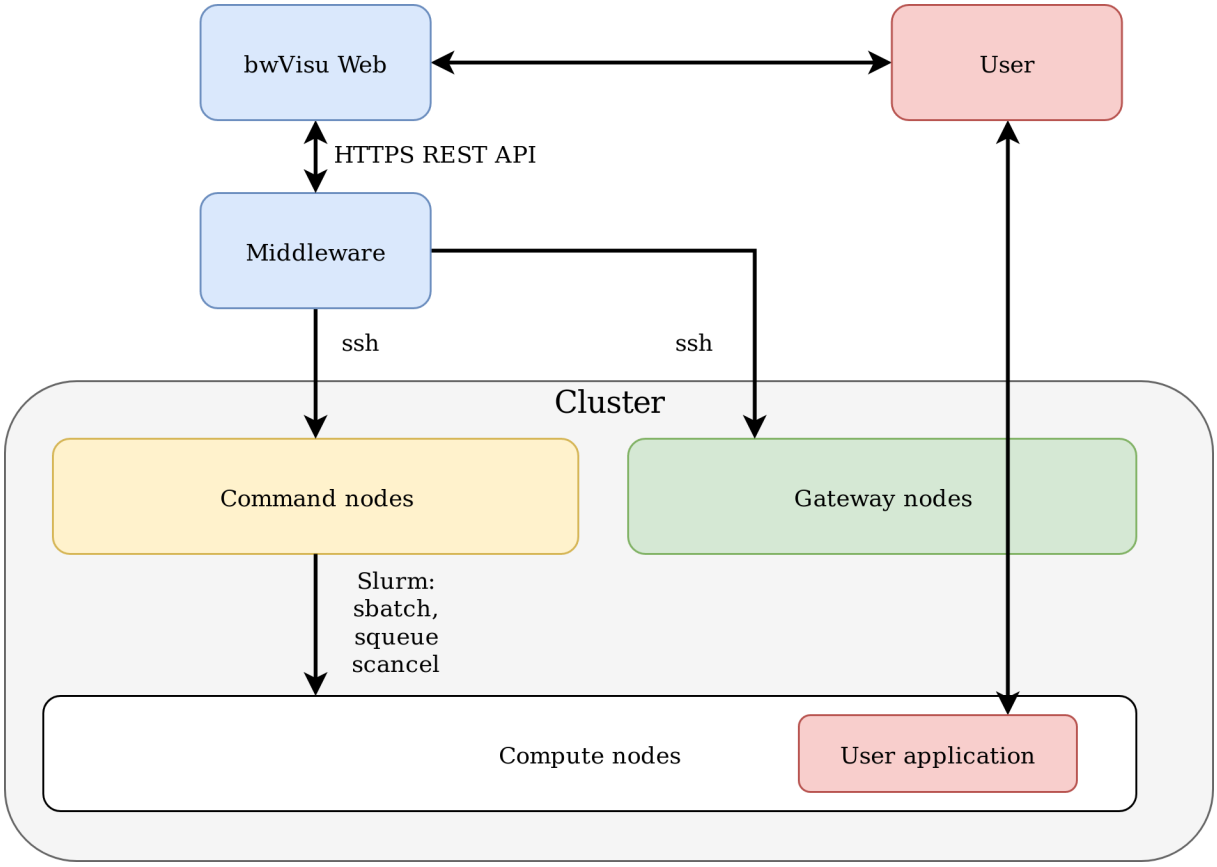

Figure 1.: The architecture of the bwVisu stack.

their web browser (Figure 2). It consists of two parts, the frontend or user interface (UI), and the backend, which supplies the frontend with data and implements the essential "business" logic (Figure 3). The user-visible part of this has only changed marginally from the mockups presented during the last E-Science-Tage conference in 2017 [20], but the architecture was revamped significantly, and the system has matured into a productive system.

While the previous system was built around Kubernetes [1], we now utilize Slurm [14] instead to allow an easier migration path from HPC clusters that previously did not use application containers. Since the latter does not provide an HTTP API that can be used by web applications to interface with it, we no longer interface directly with the HPC scheduler or orchestration engine, but use a custom-made middleware instead, as described in Section 3.

The fact that we were able to reduce the requirements for researchers to just a web browser is an important milestone made possible by the utilisation of the Xpra HTML5 client by Martin and contributors [15], which is embedded into our application container images.

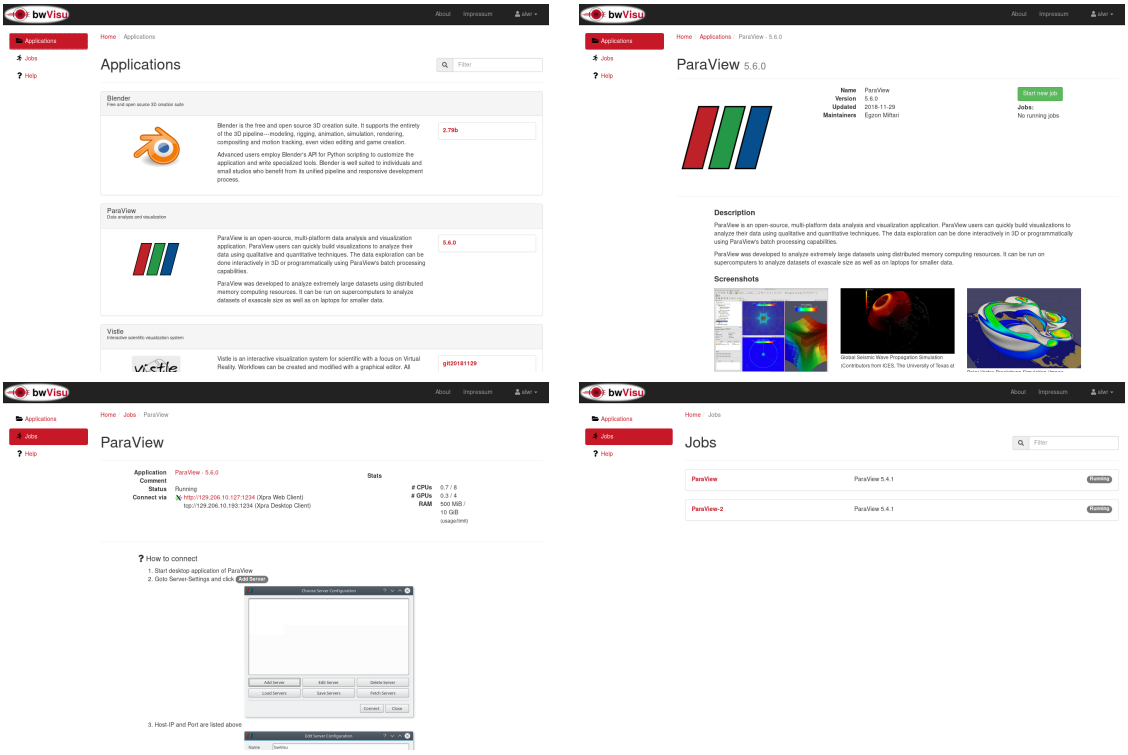

Figure 2.: bwVisu Web aims to be a user-friendly, straightforward web interface allowing users to browse available applications, learn more about them, submit them as jobs to the HPC cluster, as well as manage, connect to, and stop running jobs.

4.1. System architecture

The frontend of bwVisu Web is implemented as a light-weight single page application (SPA). "Light weight" does not only describe the low implementation complexity, but translates also to a clean visual design that is reduced to the essential functions required by users, allowing them to quickly reach their goals without distractions. The frontend application is implemented using a functional data-oriented architecture based on re-frame[2] by Thompson [22]. Rendering and updates of the HTML view is handled by Reagent[3] courtesy of Holmsand and contributors [9].

The internal interface between this frontend and the backend of the bwVisu Web system is realized using GraphQL, again popularised by Inc. [11], which defines an easy-to-use, flexible, well-defined, language-independent query language based on HTTP / WebSockets and JSON that also supports, subscription (live updates) and mutation (write access). bwVisu Web's GraphQL queries can be broken down into three categories:

1. user log in and authentication (GraphQL mutation),
2. queries for available applications and jobs currently running (GraphQL queries and subscriptions),

[2] It shares similarities with the Flux architecture popularised by Inc.[10]
[3] Which itself is based on React by Inc. [12]

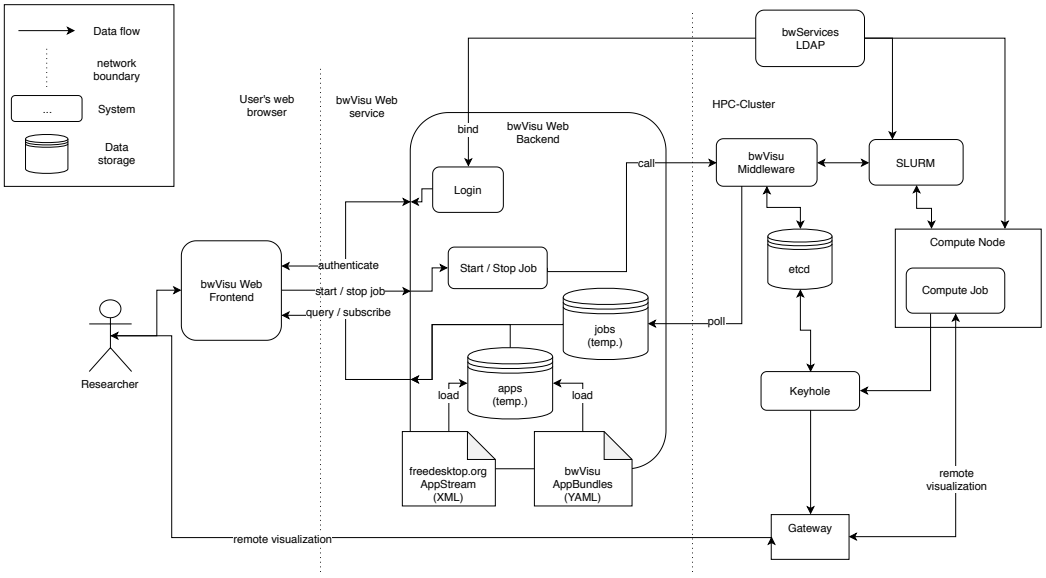

Figure 3.: bwVisu Web consists of two parts, in the user's browser and at the service provider's site, communicating via GraphQL with each other. It shares an LDAP database for user accounts with the HPC-Cluster and communicates via HTTP REST with the Infectoid middleware.

3. commands to start or stop a visualisation job (GraphQL mutations).

In exchange for a username / password pair, the authentication interface provides the bwVisu Web frontend with a JSON Web Token (JWT), which contains information about the user and will be used to authenticate all other queries. This initial log in step is performed against an LDAP database provided by the bwRegApp service developed within the project for the federal state of Baden-Württemberg [17].

The list of applications available in bwVisu is provided to the frontend as a GraphQL query and subscription (and hence supports long running user sessions). All this information is tracked in simple text files (XML or YAML), so it can easily be maintained without special tools and be stored in version control systems, e.g., directly alongside build instructions for the images themselves (e.g., `Dockerfile`s or more sophisticated build systems using `buildah`). Klumpp et al. of freedesktop.org created the AppStream 0.12 [13] infrastructure, in particular the `appdata.xml` file format, to allow developers to provide information about their applications to distributors. Using these facilities has the benefit for bwVisu that accurate and up-to-date application descriptions, screenshots and the likes are made available by the application developers themselves, which reduces the maintenance burden for bwVisu administrators. This choice poses no limitation, because this information can easily be added, should an application not provide it. In custom YAML files, which we call "appbundles", that information is extended with bwVisu-specifics, like available version numbers, corresponding container image names, and information needed by the bwVisu Web middleware in order to submit jobs to the HPC scheduler. Not merging these two into one file provides the benefit of separating

concerns and allows an easier intake of (updated) information from upstream application developers.

In the bwVisu Web frontend, this application information is used to present the list of available applications and their versions/flavours to the user, as well as to show them details, descriptions, and screenshots of individual applications. Finally, it also defines the possible settings users can define when they want to start applications and submit their jobs. These settings are being send to the bwVisu Web backend, where they are merged with any immutable data (e.g., the image name), before being passed on to the middleware. Basic structural conformance checks will be carried out in each of these steps, but actual authorization is done via user account file system permissions by the operating system of the compute nodes when the job actually starts.

Information about currently running jobs is acquired directly from the bwVisu middleware using polling and passed on to the bwVisu Web frontend via GraphQL queries and subscriptions, just like application data. bwVisu Web is equipped to handle dynamic information, such as network addresses, which are only known after job submission, when the application actually starts. Any changes compared to the previous architecture are encapsulated by bwVisu Web and should be opaque to the user.

The only external systems bwVisu Web interfaces with are an LDAP service containing the user database and the bwVisu middleware that abstracts the HPC scheduler. Additionally we avoid any persistent state within bwVisu Web itself (e.g., job databases or user session data), which follows the design proposed by Moseley and Marks in their paper "Out of the Tar Pit" [16]. We confirm their findings that a stateless design simplifies reasoning about the system, which in turn leads to less bugs and increased developer productivity.

4.2. Deployment

bwVisu Web is meant to be easily deployed onto the target HPC cluster. For this reason we selected deployment methods that can be adapted to different target systems. We utilise *Packer* by HashiCorp to build virtual machine images on common platforms like *OpenStack*, which is e.g. used by the Heidelberg *heiCLOUD* platform. The flexibility of Packer allows us to easily build images for other systems, including QEMU/libvirt for consumption on individual non-cloud machines. These virtual machine images are then deployed onto the *OpenStack* platform using *Terraform*, also maintained by HashiCorp. Terraform also allows us to target bare-metal machines and many other types of environments.

5. Visualization Applications

The applications provided in bwVisu are containerized using the well-known container platform Docker and are stored as images in a Docker Container Registry integrated into GitLab. Furthermore the CI/CD service provided by GitLab automates software development and deployment. Since bwVisu builds on well-established HPC technologies and was developed with HPC environments in mind, there is an additional conversion

step of the Docker images to the HPC-focused container platform Singularity. Unlike Docker containers, Singularity containers do not allow privilege escalation inside of them. We provide developers with base images to ensure applications are working properly and all the required dependencies are present at runtime. These base images include MPI and drivers for full hardware acceleration on Nvidia and AMD graphics nodes. Each application image is uniquely labeled and can be archived by the user, thus increasing the degree of reproducibility and portability. In addition, developers will be able to restrict access and visibility of their applications on bwVisu Web if needed. bwVisu already offers a wide range of visualization applications such as ParaView, VMD, VisIt, Blender, Vistle, and many more.

5.1. FTLE-Based Flow Visualization

We exemplify flow visualization of time-dependent flow fields in bwVisu by means of the finite-time Lyapunov exponent (FTLE). The FTLE field [6] measures the divergence of time-dependent trajectories with respect to finite-time advection, and is able to reveal the topology of time-dependent vector fields, i.e., their overall structure with respect to finite-time transport.

Given a time-dependent vector field $\mathbf{u}(\mathbf{x}, t) \in \mathbb{R}^n$ defined at each point $\mathbf{x} \in \Omega \subseteq R^n$ of the domain Ω, a trajectory (i.e., a pathline) is given as the solution to the initial value problem

$$\boldsymbol{\xi}_{\mathbf{x}_0}^{t_0}(t) := \mathbf{x}_0 + \int_{t_0}^{t} \mathbf{u}\left(\boldsymbol{\xi}_{\mathbf{x}_0}^{t_0}(\tau), \tau\right) d\tau, \tag{0.1}$$

with $\mathbf{x}_0$ being the seed point of the pathline, t_0 its seeding time, and t its integration time. From this, the *flow map* is obtained by seeding a pathline at each point $\mathbf{x}$ at time t_0, integrating it for time T, and storing the coordinates of the endpoint of the respective pathline in the map:

$$\boldsymbol{\phi}_{t_0}^{T}(\mathbf{x}) := \boldsymbol{\xi}_{\mathbf{x}}^{t_0}(T). \tag{0.2}$$

Consequently, pathline divergence can be quantified by the gradient of the flow map, leading to the definition of the FTLE:

$$\sigma_{t_0}^{T}(\mathbf{x}) := \frac{1}{|T|} \ln \|\nabla \boldsymbol{\phi}_{t_0}^{T}(\mathbf{x})\|, \tag{0.3}$$

with $\|A\|$ being the spectral norm of matrix A, i.e., the largest eigenvalue of $A^\top A$.

Apparently, the computation of the FTLE field $\sigma_{t_0}^{T}(\mathbf{x})$ is very costly, since integration of a pathline is required for each spatial sample $\mathbf{x}$, and also for each seeding time t_0 (for time-dependent FTLE visualizations, t_0 is varied to produce animations). Even worse, the structure in the FTLE field become more detailed as the advection time T is increased, requiring very high spatial resolutions. On the other hand, the computation of the pathlines is straightforward to parallelize due to their independent computation. As such, FTLE-based visualization is a perfect candidate for demonstrating and evaluating

bwVisu, also because the resulting FTLE data is a scalar field, which is, compared to the huge amount of trajectory data, comparably small. Common approaches to FTLE visualization are direct volume rendering, and extraction of ridge surfaces [5]. In this example, we follow both approaches. We distribute the entire time-dependent vector field to all compute instances, but each instance computes only a part of the FTLE field, i.e., we employ domain decomposition of the FTLE field domain. While parallel volume rendering by bricking (i.e., of the FTLE parts) is somewhat involved, ridge extraction lends itself well for distributed computation. That is, each compute instance extracts ridges from the FTLE part it computed, and sends the result to the visualization entity for interactive exploration.

We start our exploration with a time-dependent vector field obtained from a heat-driven convective 3D ($n = 3$) flow simulation in a closed container. For overview, we have a look at the velocity magnitude at time $t_0 = 5.4\,\mathrm{s}$, and vortex core lines [21], as well as instantaneous streamlines at time $t_0 = 5.4\,\mathrm{s}$ and pathlines seeded at time $t_0 = 5.4\,\mathrm{s}$ and integrated for $T = 0.15\,\mathrm{s}$ (Figure 4). Of course, such basic visualization could be accomplished without parallelization, and as such provides only limited insights in the time-dependent dynamics of this flow.

Our FTLE-based visualization, on the other hand, captures the full time-dependent dynamics. Figure 5 shows the volume rendering of the resulting FTLE field, the height ridge surfaces extracted from the FTLE field, as well as the height ridges surfaces together with a subset of the pathlines that have been used for FTLE computation, to provide flow dynamics context. One can observe that the FTLE ridge surfaces indeed separate regions of qualitatively different flow dynamics and thus provide a topological overview.

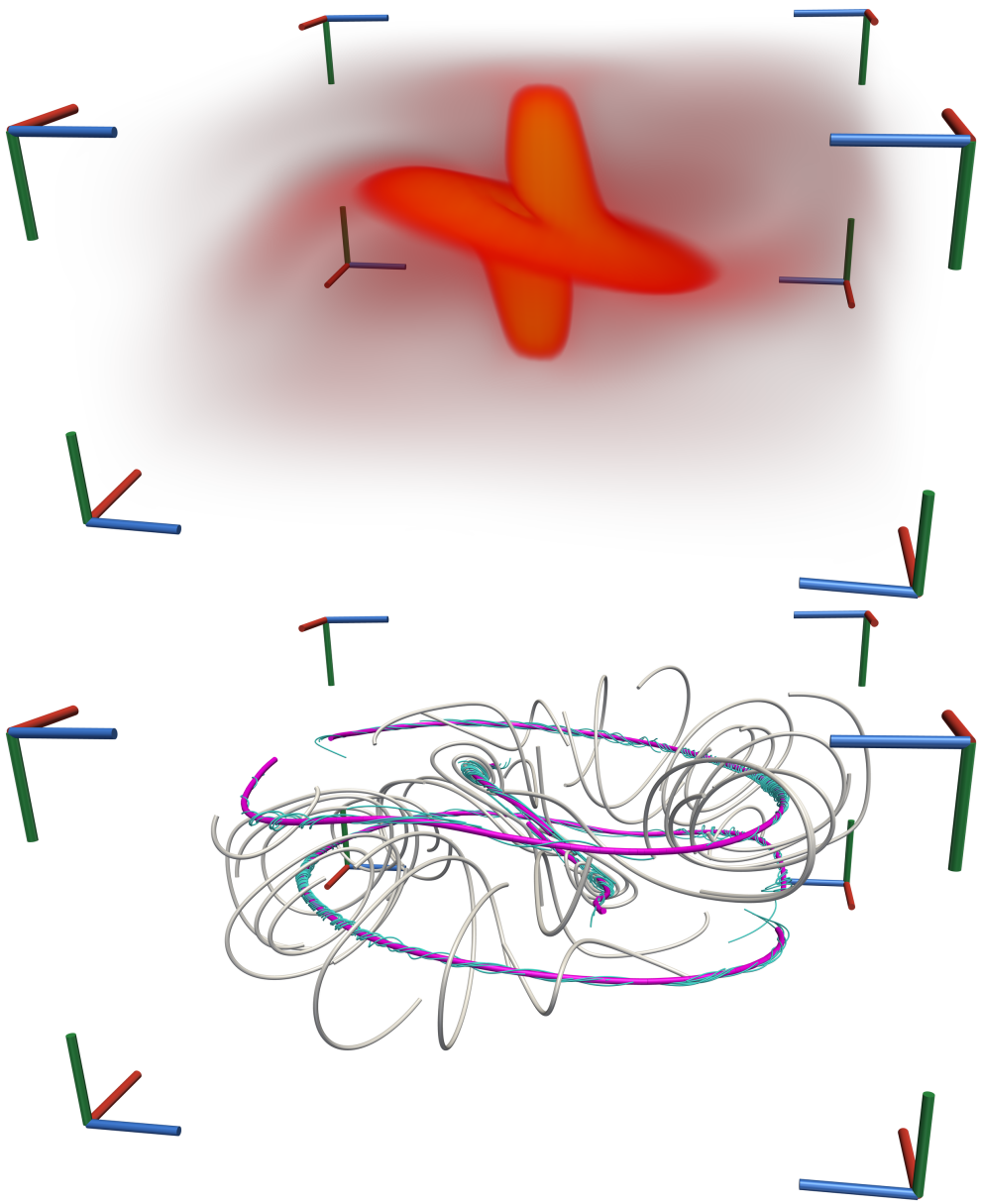

Figure 4.: Overview of Convective Flow example, at time 5.4 s. Top: Velocity magnitude by volume rendering (higher magnitude by higher opacity / brighter colors). Bottom: Vortex core lines (purple), with instantaneous streamlines along vortex core lines (blue), and pathlines seeded at $t_0 = 5.4\,\mathrm{s}$ integrated for $T = 0.15\,\mathrm{s}$ (white).

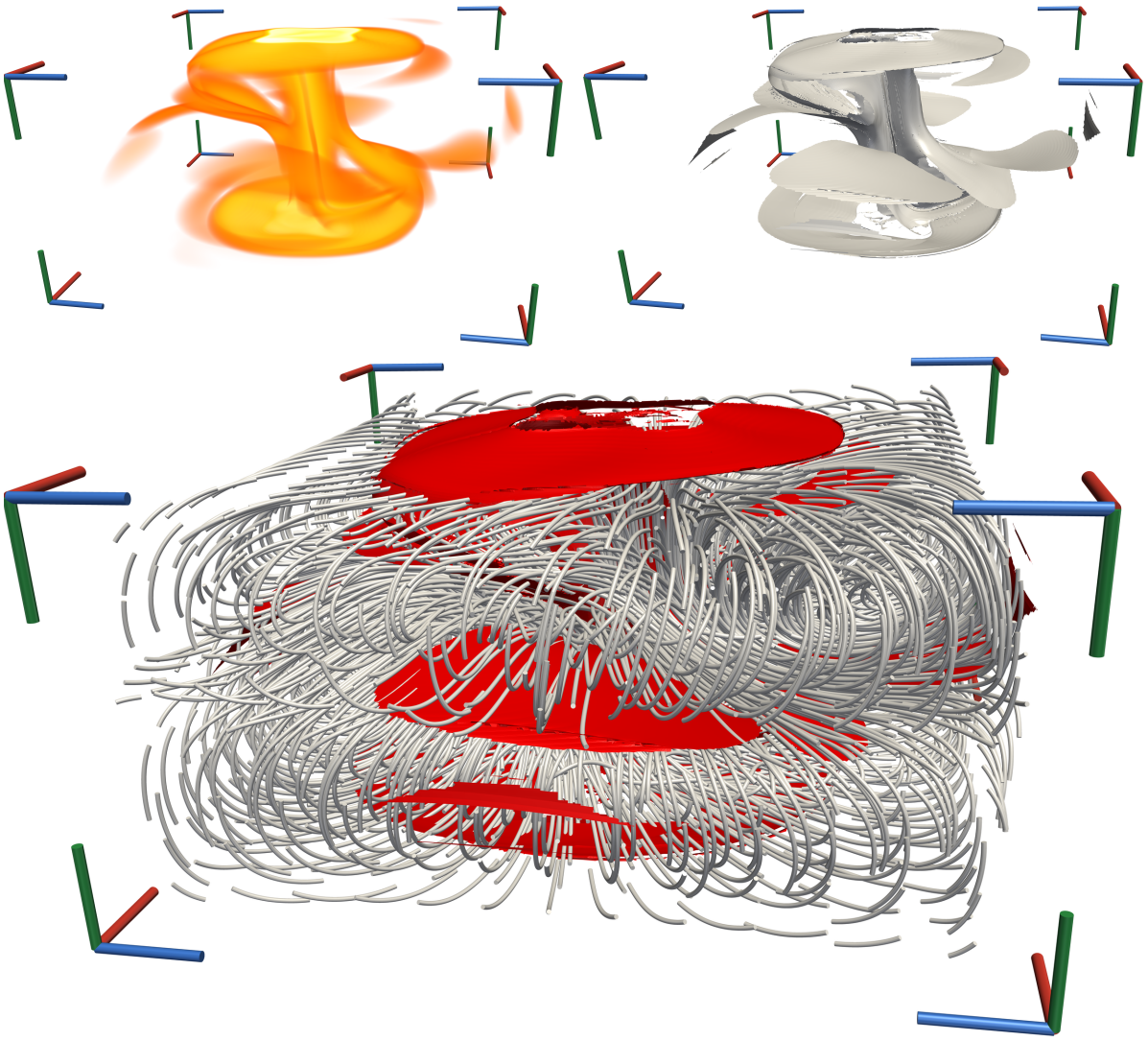

Figure 5.: FTLE-based visualization of Convective Flow example, with $t_0 = 5.4\,\mathrm{s}$ integrated for $T = 0.05\,\mathrm{s}$. Top left: Volume rendering of the FTLE field. Top right: Height ridge surfaces extracted from the FTLE field. Bottom: Height ridges with some of the trajectories used for FTLE computation, for space-time context.

6. Conclusion

We presented bwVisu, a scalable service for remote parallel visualization. We described the underlying hardware, the middleware, which has been designed with a particular focus on ease of use, the web frontend that enables effective and simple visualization, and exemplified its usage at the example of time-dependent flow visualization using the finite-time Lyapunov exponent.

Bibliography

[1] The Kubernetes Authors. Kubernetes. Production-Grade Container Orchestration. Cloud Native Computing Foundation and The Linux Foundation. URL: https://kubernetes.io/ (visited on 04/10/2019).

[2] Software: The OpenStack Authors. OpenStack. URL: https://www.openstack.org/ (visited on 04/10/2019).

[3] Martin Baumann et al. "SDS@hd – Scientific Data Storage". In: Universität Tübingen, 2017. doi: 10.15496/publikation-25204. URL: http://dx.doi.org/10.15496/publikation-25204.

[4] Software: Heidelberg University Computing Centre. heiCLOUD. Infrastructure-as-a-Service.URL: https://heicloud.uni-heidelberg.de/ (visited on 04/10/2019).

[5] David Eberly. Ridges in Image and Data Analysis. Computational Imaging and Vision. Kluwer Academic Publishers, 1996.

[6] G. Haller. "Distinguished material surfaces and coherent structures in three-dimensional fluid flows". In: Physica D 149 (2001), pp. 248–277.

[7] Software: HashiCorp. Packer. URL: https://www.packer.io/(visited on 04/10/2019).

[8] Software: HashiCorp. Terraform. URL: https://www.terraform.io/ (visited on 04/10/2019).

[9] Dan Holmsand and Reagent contributors. Reagent. Minimalistic React for ClojureScript. URL: https://reagent-project.github.io/ (visited on 04/10/2019).

[10] Facebook Inc. Flux. Application Architecture for Building User Interfaces. Facebook Inc. URL: https://facebook.github.io/flux/(visited on 04/10/2019).

[11] Facebook Inc. GraphQL. A query language for your API. Facebook Inc. url: https://graphql.org/ (visited on 04/10/2019).

[12] Facebook Inc. React. A JavaScript library for building user interfaces. url: https://reactjs.org/ (visited on 04/10/2019).

[13] Matthias Klumpp et al. AppStream 0.12. Infrastructure for distro-agnostic software-centers and universal software component metadata.https://www.freedesktop.org/ software/appstream/docs/ (visited on 04/10/2019).

[14] SchedMD LLC. Slurm workload manager.url:https://slurm.schedmd.com/ (visited on 04/10/2019).

[15] Software: Antoine Martin and Xpra contributors. Xpra. multi-platform screen and application for- warding system. URL: https://xpra.org/ (visited on 04/10/2019).

[16] Ben Moseley and Peter Marks. "Out of the Tar Pit". In: Software Practice Advancement (SPA) 2006 (Feb. 6, 2006).

[17] bwHPC project. bwRegApp. bwForCluster registration application. url:https://wiki.bwhpc.de/e/BwForCluster_User_Access#Personal_registration_at_bwForCluster(visited on 04/10/2019).

[18] The flask project. flask – a microframework for python. url:http://flask.pocoo.org/ (visited on 04/10/2019).

[19] Sabine Richling, Martin Baumann, and Vincent Heuveline. "bwForCluster MLS & WISO". In: Proceedings of the 3rd bwHPC-Symposium. Heidelberg: heiBOOKS, 2017. isbn: 978-3- 946531-70-8. doi: 10.11588/heibooks.308.418.10

[20] Dennis Schridde, Martin Baumann, and Vincent Heuveline. "Skalierbare und flexible Arbeitsumgebungen für Data-Driven Sciences". In: E-Science-Tage 2017: Forschungsdatenmanagen. Ed. by Jonas Kratzke and Vincent Heuveline. Heidelberg: heiBOOKS, 2017, pp. 153–166. isbn: 978-3-946531-75-3. doi: 10.11588/heibooks.285.377.

[21] David Sujudi and Robert Haimes. "Identification of swirling flow in 3-D vector fields". In:12th Computational Fluid Dynamics Conference. 1995, p. 1715.

[22] Michael Thompson. Re-Frame: A Reagent Framework For Writing SPAs, in Clojurescript. Mar. 2015. doi: 10.5281/zenodo.801613.