SciServer Compute: Bringing Analysis Close to the Data

Dmitry Medvedev
Johns Hopkins University
Baltimore, MD, USA
dmedv@jhu.edu

Gerard Lemson
Johns Hopkins University
Baltimore, MD, USA
glemson1@jhu.edu

Mike Rippin
Johns Hopkins University
Baltimore, MD, USA
mike.rippin@jhu.edu

ABSTRACT

SciServer Compute uses Jupyter notebooks running within server-side Docker containers attached to large relational databases and file storage to bring advanced analysis capabilities close to the data. SciServer Compute is a component of SciServer, a big-data infrastructure project developed at Johns Hopkins University that will provide a common environment for computational research.

SciServer Compute integrates with large existing databases in the fields of astronomy, cosmology, turbulence, genomics, oceanography and materials science. These are accessible through the CasJobs service for direct SQL queries. SciServer Compute adds interactive server-side computational capabilities through notebooks in Python, R and MATLAB, an API for running asynchronous tasks, and a very large (hundreds of terabytes) scratch space for storing intermediate results. Science-ready results can be stored on a Dropbox-like service, SciDrive, for sharing with collaborators and dissemination to the public. Notebooks and batch jobs run inside Docker containers owned by the users. This provides security and isolation and allows flexible configuration of computational contexts through domain specific images and mounting of domain specific data sets.

We present a demo that illustrates the capabilities of SciServer Compute: using Jupyter notebooks, performing analyses on data selections from diverse scientific fields, and running asynchronous jobs in a Docker container. The demo will highlight the data flow between file storage, database, and compute components.

CCS Concepts

•Information systems → Computing platforms;
•Computer systems organization → Distributed architectures; •Applied computing → *Astronomy*; Genomics;

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

SSDBM '16 July 18-20, 2016, Budapest, Hungary

© 2016 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-4215-5/16/07.

DOI: <http://dx.doi.org/10.1145/2949689.2949700>

Keywords

applied computing; data analysis; virtualization; application containers; scientific workflows; Docker; Jupyter; Python

1. INTRODUCTION

SciServer is a big-data infrastructure project at Johns Hopkins University that is developing a modular and scalable infrastructure for the storage, access, query and processing of large, petabyte scale, scientific datasets.

The system builds off an existing infrastructure in the astronomy domain that has been in production for the past 15 years to serve up the Sloan Digital Sky Survey (SDSS) data. This includes a number of components already, most notably the SkyServer web application, and the CasJobs batch query system. SciServer is taking these core features, extending them, augmenting them, and developing the system to support a wide range of different scientific domains that need to support very large datasets.

The project has several major goals. The first goal is to be able to access and process these large data sets in ways that would be impossible to do so by downloading the data. The second major goal is to provide personalized storage space for users of the system to store both the results of queries in database tables, and the results of additional processing in file storage, and to be able to cross-query and process their own datasets with the large petabyte scientific repositories themselves. The third goal is to provide a collaborative sharing environment in which users can share their data sets with others, either individually or in user-defined groups. The final major goal is to provide a scalable compute capability for processing and analysis of both the large data repositories and users own data. A meta objective of the overall NSF DIBBs program which is funding SciServer is to develop the system with a focus on re-usable and interoperable building blocks, using open standards were necessary, developing public API's, and being configurable to a wide range of deployment conditions and scenarios.

This paper focuses on the development of SciServer Compute, which provides a scalable environment for executing custom analyses on the server side, against both the large institutional repositories and the users own datasets. This extends the current capabilities for ad-hoc query access on databases to support Python, R and MATLAB scripting to further process the data. The newly developed open APIs for the other SciServer components, notably CasJobs, allow complex query and processing pipelines to be developed in ways that were not previously

Figure 1: Architecture overview

possible. To support this server side execution of queries and analysis, SciServer provides a large working storage area, currently hundreds of terabytes, but which will be scaled up to petabytes, to store and enable analyses on intermediate datasets that may be tens or hundreds of terabytes in size.

We have developed the first implementation of SciServer Compute that executes scripts in interactive Jupyter notebooks running within Docker containers. We are currently developing a batch mode scheduling framework for non-interactive asynchronous query and processing, which extends on CasJobs existing asynchronous framework for querying. This paper presents the work done on the interactive Jupyter notebook cluster implementation, as well as presenting a number of real-world use cases that have been developed in collaboration with partners in astronomy, cosmology, materials science, turbulence and genomics.

2. SYSTEM OVERVIEW

The goal of SciServer Compute is to provide users with their own computational workspaces in which to run their code for data analysis. Operating-system-level virtualization, or so-called containerization, offers an elegant way to create an isolated user environment, and it becomes even easier with Docker, which provides a high-level API for container management on Linux systems. Docker has already been successfully used in a number of multi-user systems running Jupyter notebooks. One of the main inspirations for our own implementation was a temporary IPython notebook service, used as a demo for an article on IPython notebooks in Nature in 2014 [5]. Building on their experience, we wanted to give users more control over the life cycle and contents of their containers,

Figure 2: File storage organization

and also to provide fast access to the large scientific datasets hosted at Johns Hopkins University from within the containerized environment. We are planning to use SciServer Compute not only for running interactive notebooks, but also for running asynchronous tasks within containers, thus allowing users to build automated workflows (a work in progress). An overview of SciServer Compute architecture and its interaction with other SciServer components is shown in Fig. 1.

Docker alone is not enough to ensure scalability, because it does not easily scale across multiple nodes. We are using virtual machines as Docker host nodes. Each VM can run a limited number of containers, but new VMs can be spawned if there is a need to increase the system's capacity. Nodes, application containers, and data volumes are orchestrated by relatively simple server-side code running as part of the dashboard web application and using a global registry database. This approach provides sufficient scalability and flexibility, and we found it less cumbersome to use than enterprise-scale container orchestration solutions, like Kubernetes.

Authentication is handled by an identity service, common for all SciServer components, which allows for seamless integration of SciServer Compute with the rest of the SciServer framework. We are using OpenStack Keystone as an identity service. It provides a simple and convenient way of token-based authentication and authorization. It also simplifies integration with other OpenStack components currently used by SciServer (Swift) or planned to be used in the future (Nova, Glance). When a user creates a new container using the dashboard, or re-visits a previously created one, an access token is injected into the container and can be used to access other SciServer services without the need of authenticating the second time. All SciServer web applications, including Compute, SciDrive and CasJobs, are using the same single sign-on mechanism based on Keystone.

Each container is using a different port on the VM to expose the Jupyter notebook server endpoint. All user requests to the Jupyter servers are routed through a proxy to avoid using non-standard port numbers directly. The Jupyter Project has its own proxy implementation, *configurable-http-proxy*, which offers a remote API for

Figure 3: User dashboard

creating and deleting routing rules on the fly. Thus we can easily update proxy routes automatically, as new containers are added to the system. It is important to note that Jupyter notebooks use WebSocket connections to establish a link with the kernel. For users connecting from behind their own proxy this could be a problem if that proxy does not support WebSocket directly, so we have to use HTTPS to make sure that all HTTP requests always stay intact to avoid any potential problems.

Users can choose from several pre-configured Docker images as base for their containers. They also choose which data volumes they want to use inside their containers. There are three kinds of file storage used in SciServer Compute (Fig. 2):

- Shared data volumes. These are not user-specific and can contain large scientific data archives or additional libraries and tools intended for common use. Shared volumes are read-only.
- Persistent user volumes. Each user has his own persistent storage space which can be used for storing any kind of personal data for long-term use. The size of persistent volumes is limited to a few gigabytes.
- Scratch user volumes. Scratch space is intended for temporary storage and processing of large amounts of data. Each user has his own scratch storage space but, unlike in persistent storage, unused files in scratch space will get automatically deleted after a certain period of time.

Persistent and scratch file storage physically resides on a separate file server, connected to the Docker cluster by a high-bandwidth, low-latency network. They are attached to all containers by default. Shared data archives are also connected by the same network, thus eliminating the need of costly and time consuming data downloads over public network. All file storage volumes in SciServer compute are encapsulated in Docker volume containers, which makes it easy to attach them to application containers as external storage. The actual data folders are mounted inside VMs as NFS mounts.

The dashboard web application (Fig. 3) allows users to manage their Jupyter notebook containers: create, delete, start, and stop them as necessary; see the status of the

Figure 4: Astronomy notebook example

existing containers. One user can create several independent containers for various purposes, for example, using different base Docker images which provide different sets of pre-installed libraries and language tools.

SciServer Compute provides a RESTful API for running asynchronous tasks inside the same container infrastructure that is used for running interactive Jupyter notebooks. An example of creating a new asynchronous task through the API is given in the listing below. The result of this request will be a new task ID, which can then be used to query the execution status, retrieve the output when it is ready, or examine the error messages.

```

POST /compute/api/container HTTP/1.1
Host: scitest02.pha.jhu.edu
X-Auth-Token: 416aee7eab474...
Content-Type: application/json

{
  "ImageName": "mro_323",
  "Label": "test",
  "Cmd": [ "python ~/workspace/persistent/test.py" ],
  "Volumes": [ "sdss_das" ]
}

```

We provide Python libraries for communication with other SciServer components from the Jupyter notebooks running within SciServer Compute framework. Though these libraries are not required, they greatly simplify communication with RESTful APIs that other SciServer components provide, most notably CasJobs and SciDrive. CasJobs is a database service that can be used to store and process tabular data, utilizing the benefits of relational database technology and direct access to the SDSS data and other data catalogs [2]. SciDrive is a file storage service for saving final computation results and sharing them with other people, built on top of OpenStack Swift object/blob store [3].

3. SCIENCE DEMOS

We will demonstrate the SciServer compute infrastructure on a number of science cases from the fields of astronomy, theoretical cosmology, studies of turbulence, genomics, oceanography and materials science. The demos show the versatility of our infrastructure in its applicability

(a) Brittle materials fragmentation simulation [1]: stress field and crack propagation

(b) Pressure field from JHU TurbDB [4]: original and filtered with Gaussian convolution kernel of increasing size

Figure 5: Computation examples

to these diverse domains. We show explicitly how to access and connect the several components of the SciServer framework. We will query some of the largest scientific databases that are publicly accessible to the community and perform non-trivial analyses on their results.

Access to the components is provided through our single sign-on mechanism that is an integral part of all the REST APIs supported by these components. We show how to store results on our scratch file system and in our scratch databases. These extensions to the original CasJobs design allows users to store intermediate data sets up to several terabytes in size. We show both interactive notebooks in Python, R and MATLAB, and show how to submit jobs to our batch queues from these notebooks.

Some of the scientific analyses we perform are:

- **Cosmology:** the determination of scaling relations in the density profiles of dark matter halos extracted from cosmological n-body simulations;
- **Genomics:** the retrieval of individual gene sequences from the database underlying the Terabase Search Engine and their transformation to the SAM file format standard;
- **Materials science:** the determination of fragments and their scaling relations in simulations of the fragmentation of two-dimensional sheets of materials under tension using graph techniques (Fig. 5a);
- **Turbulence:** the filtering of gigabyte size datasets extracted from isotropic turbulence simulations (Fig. 5b);
- **Oceanography:** the visualization and analysis of stream patterns in three-dimensional simulations of ocean flows.

4. ACKNOWLEDGMENTS

SciServer is funded by the U.S. National Science Foundation Award ACI-1261715.

5. REFERENCES

- [1] N. Daphalapurkar. FragData – high fidelity data on dynamic fragmentation of brittle materials. In *IDIES Annual Symposium*. Johns Hopkins University, 2014.
- [2] N. Li and A. R. Thakar. CasJobs and MyDB: A batch query workbench. *Computing in Science and Engineering*, 10(1):18–29, January/February 2008.
- [3] D. Mishin, D. Medvedev, and A. Szalay. Data sharing and publication using the SciDrive service. In *Astronomical Society of the Pacific Conference Series*, volume 485, pages 465–468, 2014.
- [4] E. Perlman, R. Burns, Y. Li, and C. Meneveau. Data exploration of turbulence simulations using a database cluster. In *Supercomputing SC07*. ACM, IEEE, 2007.
- [5] H. Shen. Interactive notebooks: Sharing the code. *Nature*, 515:151–152, 6 November 2014.