CYBER INFRASTRUCTURE AS A SERVICE TO EMPOWER MULTIDISCIPLINARY, DATA-DRIVEN SCIENTIFIC RESEARCH

Xiangrong Ma[1], Zhao Fu[1], Yingtao Jiang[1], Mei Yang[1], Haroon Stephen[2]

[1]Department of Electrical and Computer Engineering, University of Nevada, Las Vegas, Las Vegas, USA 89154
[2]Department of Civil and Environmental Engineering and Construction
University of Nevada, Las Vegas, Las Vegas, USA 89154

ABSTRACT

In supporting its large scale, multidisciplinary scientific research efforts across all the university campuses and by the research personnel spread over literally every corner of the state, the state of Nevada needs to build and leverage its own Cyber infrastructure. Following the well-established as-a-service model, this state-wide Cyber infrastructure that consists of data acquisition, data storage, advanced instruments, visualization, computing and information processing systems, and people, all seamlessly linked together through a high-speed network, is designed and operated to deliver the benefits of Cyber infrastructure-as-a-Service (CaaS).There are three major service groups in this CaaS, namely (i) supporting infrastructural services that comprise sensors, computing/storage/networking hardware, operating system, management tools, virtualization and message passing interface (MPI); (ii) data transmission and storage services that provide connectivity to various big data sources, as well as cached and stored datasets in a distributed storage backend; and (iii) processing and visualization services that provide user access to rich processing and visualization tools and packages essential to various scientific research workflows. Built on commodity hardware and open source software packages, the Southern Nevada Research Cloud(SNRC)and a data repository in a separate location constitute a low cost solution to deliver all these services around CaaS. The service-oriented architecture and implementation of the SNRC are geared to encapsulate as much detail of big data processing and cloud computing as possible away from end users; rather scientists only need to learn and access an interactive web-based interface to conduct their collaborative, multidisciplinary, data-intensive research. The capability and easy-to-use features of the SNRC are demonstrated through a use case that attempts to derive a solar radiation model from a large data set by regression analysis.

KEYWORDS

Cyber infrastructure-as-a-Service; cloud computing; big data; Map Reduce; data-driven scientific research.

1. INTRODUCTION

Funded by the U.S National Science Foundation(NSF) since 2013, the Nevada Solar Nexus project**[1]**marks a long term, large scale, state-wide effort to study the impacts of solar energy generation on limited water resouces and fragile desert environment in the state of Nevada. One of the key elements of the Nexus project, truly multidisciplinary by nature, is to build and leverage its Cyber infrastructure(CI) to support a large array of Nexus researchers (college professors, post docs, graduate students, undergraduate students, and community participants) for their research endeavors. Up to date, these Nexus researchers have collected over 30GBytes of geospatial data (atmosphere, precipitation, soil, and vegetation) and 1TB of image data from 13 sensor towers at multiple sites, and the size of the data collected is continuously growing every day at a very fast pace. These valuable data are all delivered and deposited directly to the Nevada Research Data Center (NRDC), a data repository physically located at a university campus in northern Nevada. Except the data storage and management services offered by the NRDC, the southern Nevada is tasked to provide all the remaining cyber infrastructure services, including data integration, data mining, data analysis/processing, and data visualization as well as other needed computing and information processing services, through a private cloud, named as the Southern Nevada Research Cloud(SNRC). All the sensors and the CI services provided by NRDC and SNRC are seamlessly linked together by high-speed networks. If fully realized, this Nexus CI will drive the entire research community to take advantage of all its capabilities to pursue their data-intensive, multidisciplinary research ambitions.

Design, implementation, and operation of the Nexus CI follows the increasingly prevalent “as a service” model, hereafter dubbed as Cyber infrastructure-as-a-Service (CaaS). With CaaS as the delivery model, the SNRC is specially designed to bring traditionally separate High Performance Computing(HPC) and Big data processing platforms together to create unified cloud platform. In addition, through a web-based service endpoint, integration of SNRC and NRDC allow send users to effortlessly access and process remote large datasets through high-speed networks and advanced data processing and storage capabilities. The CaaS can be divided into three major service groups:

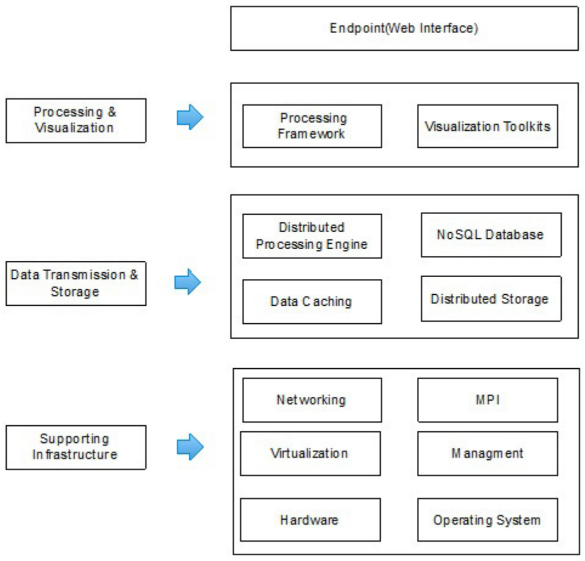


Figure1. The major service groups and servcies of the CaaS model.

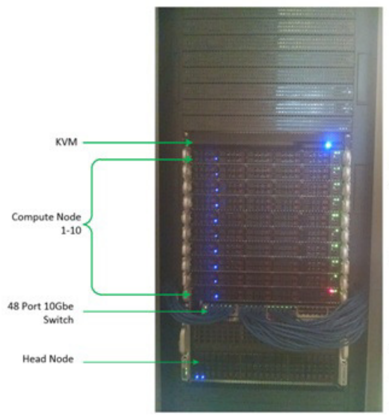
1) *Supporting Infrastructural Services* that comprise all types of hardware, operating system, management tools, virtualization, and message passing interface(MPI);
2) *Data Transmission and Storage Services* that provide the connectivity to Big data sources, as well as cached and stored datasets in a distributed storage backend;
3) *Processing and Visualization Services* that provide user an access to computing resources and tools essential to process and visualize large datasets.

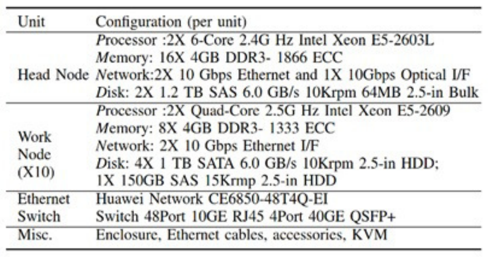
All the major services in each of the three service groups in CaaS are shown in Figure 1. End user can access all of these services through a web-based interface. Note that majority of the services are built upon open source projects, which considerably cuts down the development cost and time. The time and financial resources thus saved are rather diverted to the components unique to the Nexus CI, such as the data caching and system management. In what follows, Section 2 details the design and implementations related to the infrastructural level services, such as hardware, Operating System, virtualization and networking. Section 3 and Section 4 cover the design and implementation issues of the storage and processing subsystems, respectively. The capability and easy-to-use feature of the SNRC are demonstrated through a use case described in Section 5.Finally, the paper is summarized in Section 6.

2. SUPPORTING INFRASTRUCTURAL SERVICES

2.1. Hardware Resources

All the hardware components needed by the SNRC were determined by evaluating various factors, tradeoffs and scenarios, including cost, turn-around time, estimation and prediction of current and future data volumes, number of users to be supported, vendor reputation, and type and level of technical support from vendors. The major hardware components of the SNRC are listed in Figure 2, and they are housed in a server rack shown in the Figure.





Unit	Configuration (per unit)
Head Node	*P*rocessor :2X 6-Core 2.4G Hz Intel Xeon E5-2603L *M*emory: 16X 4GB DDR3- 1866 ECC *N*etwork:2X 10 Gbps Ethernet and 1X 10Gbps Optical I/F *D*isk: 2X 1.2 TB SAS 6.0 GB/s 10Krpm 64MB 2.5-in Bulk
Work Node (X10)	*P*rocessor :2X Quad-Core 2.5G Hz Intel Xeon E5-2609 *M*emory: 8X 4GB DDR3- 1333 ECC *N*etwork: 2X 10 Gbps Ethernet I/F *D*isk: 4X 1 TB SATA 6.0 GB/s 10Krpm 2.5-in HDD; 1X 150GB SAS 15Krmp 2.5-in HDD
Ethernet Switch	Huawei Network CE6850-48T4Q-EI Switch 48Port 10GE RJ45 4Port 40GE QSFP+
Misc.	Enclosure, Ethernet cables, accessories, KVM

Figure2. The hardware components of SNRC.

2.2. Operating System

Traditional GNU/Linux distributions have some limitations in a cloud-computing context, for reasons summarized below.

- *Compatibility:* Most traditional Enterprise Linux distributions have release a cycle spanning three to five years, depending on certain Long-term support kernel and foundational tools and libraries (e.g. C libraries). While a new distribution is still under development, often those outstanding distributions may have already become obsolete, making them incompatible with many tools which are desired or must be supported for scientific research.
- *Unnecessary bloat:* The one-size-fits-all design tends to make the distributions unnecessarily large and bloat, which leads to an enlarged test matrix for a new release.
- *Unoptimization:* Since all the binary distributions attempt to support as many platforms as possible, they usually target the most popular micro-architectures in the market. Thus, new instructions and other less popular hardware resources are typically poorly supported, or even left out unsupported.

These limitations forced us to develop a new distribution to meet the needs of the SNRC. The new distribution was based on Gen too Linux [6], a source-based rolling release meta distribution which offers a large degree of flexibility for distribution customization and optimization. Our distribution was highly customized to include just the essentials and User land applications, while performance optimization was performed to reflect the specific processor architectures and application environment that we were working with. Security was enhanced with kernel hardening [7].The distribution already saw an average of 2% performance gain with compiler flags tuning only. Introducing faster libraries had another huge impacts on application performance; we noticed up to 36X speedup on matrix operations when choosing Open BLAS[8]over net lib[9]. However, deployment of the new distribution on physical computing nodes met some practical problems. First, this distribution experienced resistance from the system administrative stuff that did not possess the skill set and understanding of the underline operating system internals. Secondly, support of physical hardware required more device drivers and administrative tools, which were not always useful in user application environment. Instead, Centos 7 was finally a dopted to be the base OS running on the physical machines (Figure 1) upon its public release, and the new distribution was later repacked to include cloud image, running inside the virtual machines and containers.

2.3. Virtualization and Networking

Virtualization helps improve the scalability, efficiency and availability of resources. Traditional virtualization tools, such as libvirt, were found to have poor scalability from our own experimental study.

.

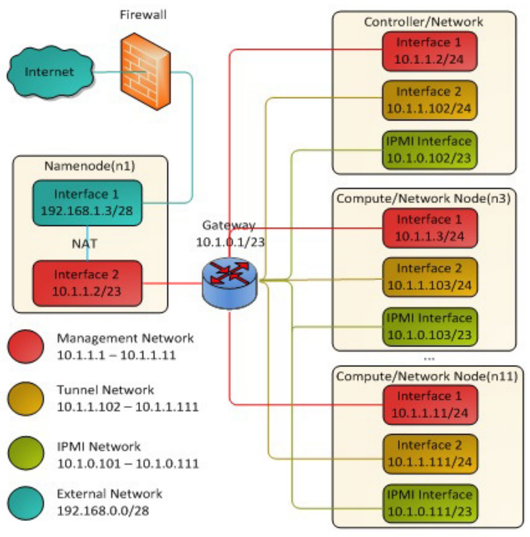


Figure 3. The Network Architecture.

As the number of virtual instances increases, the network performance could drop dramatically. In our experiment,40 virtual instances were setup by libvirt with KVM[10], but2 out of the 40 nodes suffered a packet lose rate of more than90% constantly. This serious problem was solved by adding Open Stack[11] and Open vSwitch (OVS) [12]. OVS supports standard management interfaces and protocols and enables network automation through programmatic extension. To isolate traffic between data processing and cluster control/management services, networks were segmented into three different Virtual LANs (VLANs) as illustrated in Figure3.

Figure 4 shows how traffic is routed from the North to the South (i.e., traffic between a virtual instance and the external network). Firewall and packet state tracking are handled by the security groups, which can be configured within Open Stack. To send packets from a virtual instance to the external network (the Internet in our case), firstly, the instance tap interface forwards packets to the

Linux bridge (qbr), after which the packets pass through the OVS integration bridge (br-int) port (qvb). Patch ports, a pair of virtual devices that act as a patch cable, pair the integration bridge and tunnel bridge. The tunnel bridge sends the data packets to the controller node over a physical interface using the Generic Routing Encapsulation (GRE) [13], as our payload protocols are compatible, but payload addresses are not. The controller next forwards the data packets from the tunnel bridge to the integration bridge. Since there might be multiple virtual networks, a namespace router is included. This router forwards packets from the integration bridge to its gateway (qg), which is patched with the external bridge. The external bridge sends the data packets to the physical gateway that connects to the Internet.

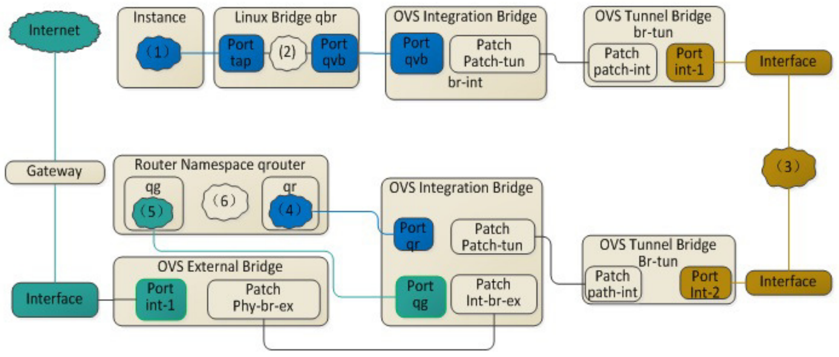


Figure 4. Network traffic flow (North-South).

2.4. Management

Not only do management tools help reduce the human labor needed, they actually are a key piece to the success of this project. Various tools have been deployed to ease the configuration and management task throughout the development. In specific, hardware resources are managed through Open Stack and IPMI tools, while users are managed through LDAP. Besides these existing tools, a number of scripts were developed in-house to automate the management tasks. The script developed for OS installation and configuration, for instance, enables the MAC address to be hashed into the hostnames and internal IP address, and then it generates Kick Start files based on the customized templates. TFTP and other services were configured to allow computing nodes to boot from the Intel Prebootе Xecution Environment (PXE), loading the OS image from the head node through the network and then completing the installation.

3. Data Transmission And Storage Services

3.1. Data Transmission

Data transmission is one central service that needs to be provided bythe CaaS. Allowing users to have seamless access to external datasets saves them huge amount of time otherwise spent on data preparation. An efficient data transmission model shall minimize redundant data transmissions and optimize the network connection query to give users best possible Quality-of-Service (QoS) experiences. In SNRC, we implemented a transmission scheme that involves resource caching and connection tracking and management, as illustrated in Figure 5.
The data transmission unit consists of five major components: REST clients, a global connection manager (CM), a caching server, a sanitizer, and persistent local storage (e.g., a database or a file system). A REST client serves as an adapter to different data sources, and provides user API to query and download datasets stored in a remote site. Acaching server saves a REST request and

its corresponding response asone key/value pair to avertredundant transmissions. The CM puts all the external network requests into a queue, schedules them based on their QoS requirements, and monitors the connection status. To service one data request, following steps are typically involved.

- The system first checks if the data are already in the local storage system. If yes, the location of that data will be immediately returned.
- If the data are found not locally available, check if it is in caching server. If it is a hit, return the data.
- Otherwise (it is a miss), send a request to the CM, and the CM will queue and execute that request. Upon fulfilling that request, the data will be sent back to the caller that initiated the request.
- The sanitizer will validate the data and store it into a local storage system.

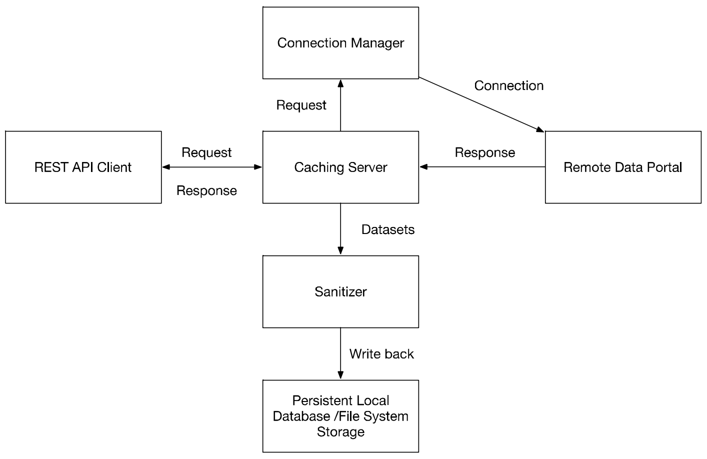


Figure5. The workflow to servcice a data loading request.

3.2. Distributed Storage Subsystem

A typical distributed storage system, like Red Hat Global File System (GFS) and Oracle Cluster File System (OCFS), segments and stores data into different blobs among the hosts, while leaving processingtoend user. A MapReduce[14] based system(Google GFS and Hadoop) is more feasible for our purposes as it seamlessly integrates storage and processing.The MapReduce model involves applying a map operation to key/value pairs, then a reduce operation to all the values sharing the same key, and a merge operation to process the result. Hadoop [15] is a framework for running applications on large clusters built from commodity hardware. It takes advantage of data locality [16] to reduce the communication cost in parallel processing. Hadoop implements the MapReduce paradigm and provides a distributed filesystem – the Hadoop Distributed File System(HDFS). However, when data are stored on a single storage cluster and shared among users, access control and QoS can be addressed by setting up two storage clusters (Figure 6): TheVirtual Storage Cluster(VSC) and the Physical Storage Cluster(PSC).

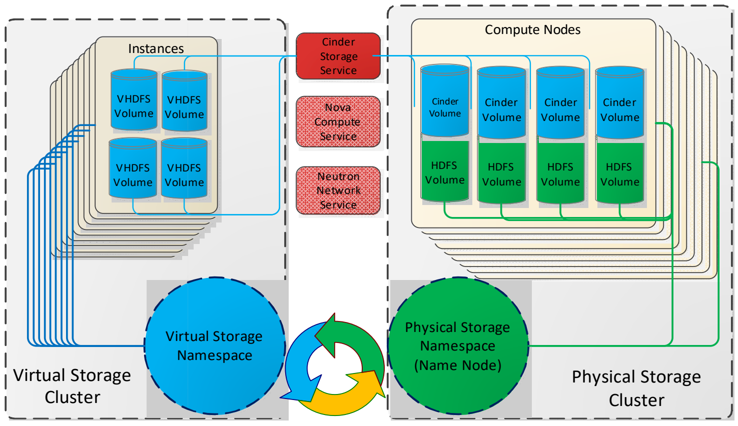


Figure6. Storage System Design

VSC resides in a virtual cluster managed by the OpenStack, and can be accessed by users. With an essentially identical architecture as VCS, PSC can provide similar functionalities, but it mainlyservices the persistent storage needs. MapReduce jobs initiated by users will be dispatched to the intermediate cluster, while streaming sensor data will be stored in the persistent cluster.Note that as high as an order of magnitude of performance penalty canbe paid if a wrong storage scheme is adopted, mandating the system to be fine-tuned to support I/O intensive applications. In this study, to maximize the I/O throughput, each physical hard drive was partitioned into two volumes: the cinder volume and the HDFS volume.

As illustrated in Figure 6, HDFS volumes are configured as a block storage device for persistent cluster using LVM, while cinder volumesare managed by the cinder (the block storage service for OpenStack), and both are exposed to the virtual instances through iSCSI (Internet Small Computer System Interface). In this context, virtual instances shouldbe built (withwritten scripts) toensure that an instancehas been granted access only to the disk in its hosted node. With respect to the diversity of the real-world scientific computational needs, the SNRC was tested in terms of network performance and distributed parallel data processing performance.Since the primary focus of SNRC is on scientific big data processing, the HiBench [18] benchmark suite was used to test overall system performance in terms of speed (i.e., job running time), throughput, I/O bandwidth. HiBench consists of a set of workloads from both synthetic and real-world applications.

In our test, four workloads(Database join, aggregation, Bayesian classification and K-means clustering) were selected to cover typical application scenarios in scientific data processing, three popular micro benchmarks were included, and two HDFS benchmarks (DFS read and write) were used for assessing the system performance of the HDFS.

Table I Distributed Processing Perforamnce

	Data Size(GB)	Duration (sec)	Total Through-put
sort	3.29	35.44	92.68MB/s
terasort	32.00	116.62	274.40 MB/s
wordcount	32.84	170.08	193.14 MB/s
dfsioe-read	27.00	91.76	294.22 MB/s
dfsioe-write	27.15	108.11	251.17 MB/s
aggregation	0.32	47.97	6.67 MB/s
join	1.83	97.52	18.81 MB/s
scan	1.75	33.72	51.95 MB/s
bayes	1.88G	363.94	5.17 MB/s
kmeans	20.08G	336.50	59.67 MB/s

Tests were performed on the physical storage cluster among 10 data nodes, and results were tabulated in Table I. The HDFS workload test showed that the throughput of HDFS could reach up to 294 MB/s on read, and 251 MB/s on write.When dealing with the work load of “sort” (Table 1), the system could process a total of 3.29GB of data in about 35 seconds, a mere throughput of 92.68 MB/s. When the better map-reduce-based algorithms were used, the TeraSort could process 32GB of data in only 116 seconds, boosting the throughput to 274.40 MB/s, which is close to DFS I/O peak throughput.

4. Processing And Visualization

The multidisciplinary research flavor in Nexus requires diverse tool and programming language support. For decades, scientists have been using imperative programming language such as C, C++ or Fortran to code and run their scientific models ,after which they submit their programs to a place like Portable Batch System(PBS); parallelism is achieved by using multi-thread and message passing library. As the programming paradigm shifts, declarative programming languages, such as Matlab/Octave and R, have gained popularity. They offer comparable performance but demand much shorter development time.

As the world is going to rely on more distributed storage, it puts additional burden on users to acquire deeper understanding of Map Reduce and even require them to learn a both new and old language: Java. The Resilient Distributed Datasets (RDD) abstract parallel data structures [17]allow users to explicitly persist intermediate results in memory, encapsulate implementation details away from users, and offer a rich set of operators for data processing. Spark [18], based on the concept of RDD, is easier to program and performs better when all the data can fit into the memory; it also performs real-time processing using the existing machine-learning libraries and other toolboxes.

The IPyth on Notebook (now known as the Jupyter Notebook) can integrate multiple computing kernels into one single computational environment, combining code execution, rich text, plots and rich media into an interactive document. The notebooks servers can be deployed in hypervisor-based VM or Linux Container (e.g., docker) with near to native performance. These processing framework and tools (and many others) were bundled together and integrated into our cloud distribution. Figure 7 shows the organization of these components. From a remote device, a user logs into the notebook system, calling a REST client to import data from a remote site into the HDFS. The data are saved in the HDFS and cached as a RDD in memory using Spark. The RDD can be processed in Octave, R, or Python using any machine learning libraries or other packages, final and immediate results can all be visualized within the notebook instantly.

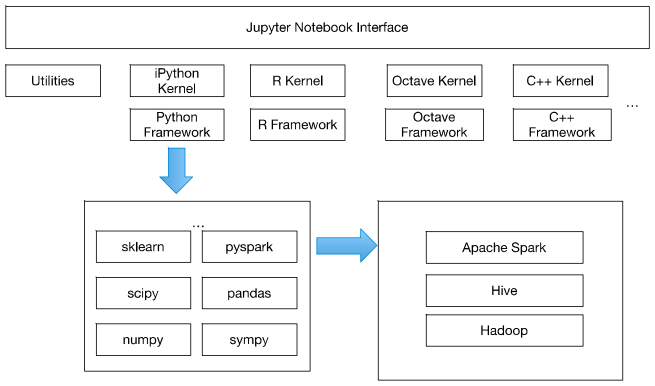


Figure7. Theprocessing and visualization framework.

5. Usecase Study: Building A Solar Radiation Model

The objective of this use case is to showcase how to build a solar radiation model that can guide the allocation of solar energy resources in different geographical locations of Nevada at different times. This model will be built upon from the Geospatial datasets collected from the 13 sensor stations over six years of time,stored in NRDC, and processed using the SNRC platform.

5.1. Data Preparation and Analysis

There are 2236 measurementsperformed every minute, bringing a total of 7 billionplus data records in the dataset. These data can be transferred from the NRDC to the SNRC Jupyter notebook with the data transmission API given below:

```
napi.getcsvfile('2012-01-01 0:01 AM', '2017-01-01 0:01 AM', sensor_list,"solardata.csv",)
```

Since our focus is limited to solar radiation measured in photovoltaics (PV) panels, all irrelevant data records would have to be purged. After purging, the resulting CSV file would contain all the solar radiation data of the past six years, with the first nine rows as the headers, followed by rows of numerical values arranged according to their timestamp indices. Pandas library provides a two-dimensional heterogeneous tabular data structure named as Data Frame[19], and arithmetic operations can be performed on both rows and columns. Note that most of the functions provided by Pandas' Data frame have their equivalentsin Spark, which is also supported by SNRC.

The downloaded CSV file can be parsed into Data Frame by running the following:

```
dataset = pandas.read_csv("./solardata.csv", sep=','…)
```

In our case, parsing the downloaded file (2.2 GB) into the Data Frame took less than 1 minute to complete. We chose to study monthly mean radiation at 12:00 AM (Pacific Time),which involved down-sampling of the radiation data. Since the time series data are indexed by their timestamps, we can locate values using the Data built-in functions provided by pandas, and have such data later converted to a dictionary data structure with year as its key.

```
hourly_mean  =  ( dataset['Mojave'].groupby(TimeGrouper(freq='H')).aggregate(np.mean)  )
grouped  =  (hourly_mean.between_time("20:00","20:00").groupby(TimeGrouper(freq='M'))
                  .aggregate(np.mean).groupby(TimeGrouper(freq='365d')))
d = {K.year: group.reset_index () ['Mojave'] for K, group in grouped if K.year< 2017}
```

Now we can visualize the processed data in bar diagram as shown in Figure 8.a

```
# Label and Ticks settings
pandas.DataFrame (d).plot (kind='bar', ax=ax, width=width)
```

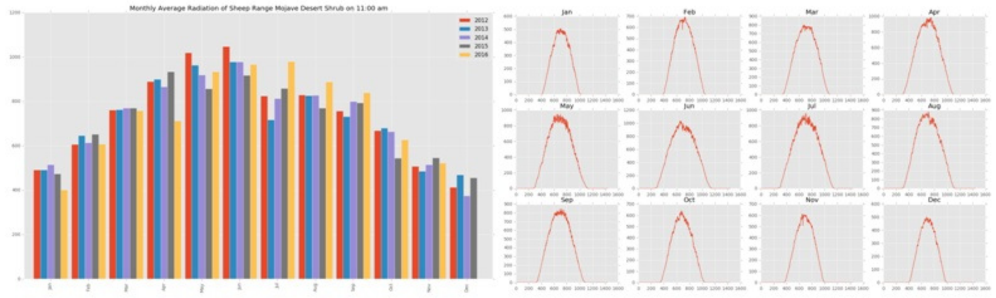


Figure 8. a) montly average solar radiation, and b) daily radiation by month.

5.2. Model Curve Fitting

Figure 8.b helps scientists tovisualize how solar radiation varies with the hourson a day. A theoretical study would generate a physics-backed model to explain the relationship between the solar position and the reflect rate. On the other hand, since we already have enough observed data, one can quickly derive and fine tune a numerical model using these observed data. Simple inspection of the data plotted in Figure 8.bcan lead to a hypothesis that the solar radiation maintains a sinusoidal relationship with respect to time. That is,

$$Y = \begin{cases} A\sin(2\pi BX + C) + D, if \sin(2\pi BX + C) \geq 0 \\ 0\ otherwise \end{cases}$$

where X is the timestamp in minutes, Y is the radiation, and A, B, C, D are parameters to be estimated. These parameters, for instance, can be easily obtained with least square approximation by calling a function from the Notebook.

```
Opt_func = lambda t: model(x, t) -y
a,b,c,d = leastsq(opt fun, guessed Para)
```

The regression model thus obtained can be readily compared with the observed data (Figure 9) by plotting them together in Figure 9. Apparently, this regression model fits the observed data really well.

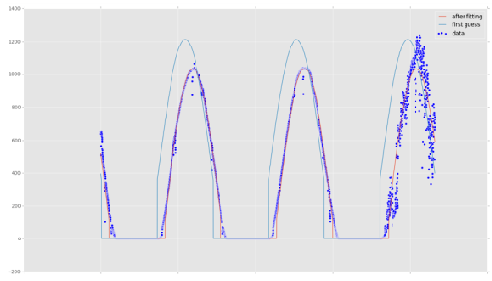


Figure 9. Results obtained from the regression model are compared against the observed data.

6. SUMMARY

This paper described various services and features of the SNRC, a platform specially developed to support a large research team engaging in multidisciplinary, data-driven research efforts.

Organized around the delivery model of cyber infrastructure-as-a-service (CaaS), the SNRC currently can store and process tens of Tera Bytes of data and support over 100 researchers scattered around multiple campuses and remote sites. Capability of SNRC can be easily scaled up to process hundreds to thousands Tera Bytes of data and support significantly more users, with installation of additional commodity hardware resources.

REFERENCE

[1] Nvsolarnexus.org, “The Solar-Energy-Water-Environment Nexus Project,” [Online]. Available: http://nvsolarnexus.org.

[2] S. Dascalu, F. C. Harris Jr, M. McMahon Jr, E. Fritzinger, S. Strachan, and R. Kelley, “An Overview of the Nevada Climate Change Portal,” Proc. 7th International Congress on Environmental Modelling and Software (iEMSs), 2014, vol. 1, no. 2014, pp. 75–82.

[3] V. D. Le, M. M. Neff, R. V Stewart, R. Kelley, E. Fritzinger, S. M. Dascalu, and F. C. Harris, “Microservice-based architecture for the NRDC,” Proc. IEEE International Conference on Industrial Informatics(INDIN), 2015, pp. 1659–1664.

[4] G. Foundation Inc, “Gentoo Linux Project,” [Online]. Available: http//www. gentoo. org.

[5] A. Chuvakin, “Linux Kernel Hardening,” [Online]. Available: http://www.symantec.com/connect/articles/linux-kernel-hardening.

[6] Z. Xianyi, W. Qian, and Z. Chothia, “OpenBLAS” [Online]. Available: http//xianyi. github. io/OpenBLAS.

[7] S. Browne, J. Dongarra, E. Grosse, and T. Rowan, “The Netlib mathematical software repository,” D-Lib Magazine, vol. 1, no. 3 Sep. 1995.

[8] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, “KVM: the Linux Virtual Machine Monitor,” Proc. Linux Symposium, 2007, vol. 1, pp. 225–230.

[9] O. Sefraoui, M. Aissaoui, and M. Eleuldj, “OpenStack: Toward an Open-Source Solution for Cloud Computing,” International Journal of Computer Applications, vol. 55, no. 3, pp. 38–42, Oct. 2012.

[10] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, and others, “The design and implementation of open vswitch,” Proc. 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2015, pp. 117–130.

[11] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina, “Generic routing encapsulation (GRE),” RFC 2784, Mar. 2000.

[12] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” Communications of the ACM, vol. 51, no.1, pp. 1–13,Jan, 2008.

[13] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The Hadoop distributed file system,” Proc. 26th IEEE Symposium on Massive Storage Systems and Technologies, 2010.

[14] A. Rogers and K. Pingali, "Process decomposition through locality of reference," ACM SIGPLAN Notices, vol. 24, no. 7, pp. 69–80, Jul. 1989.

[15] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, “Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing,” Proc. 9th USENIX conference on Networked Systems Design and Implementation, 2012, pp. 2-2.

[16] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang, “The HiBench benchmark suite: characterization of the MapReduce-based data analysis,” Proc. 26th International Conference on Data Engineering Workshops, Mar. 2010.

[17] http://pandas.pydata.org/pandas-docs