Supporting distributed, interactive Jupyter and RStudio in a scheduled HPC environment with Spark using Open OnDemand

Jeremy W. Nicklas
Ohio Supercomputer Center
Columbus, Ohio
jnicklas@osc.edu

Doug Johnson
Ohio Supercomputer Center
Columbus, Ohio
djohnson@osc.edu

Shameema Oottikkal
Ohio Supercomputer Center
Columbus, Ohio
soottikkal@osc.edu

Eric Franz
Ohio Supercomputer Center
Columbus, Ohio
efranz@osc.edu

Brian McMichael
Ohio Supercomputer Center
Columbus, Ohio
bmcmichael@osc.edu

Alan Chalker
Ohio Supercomputer Center
Columbus, Ohio
alanc@osc.edu

David E. Hudak
Ohio Supercomputer Center
Columbus, Ohio
dhudak@osc.edu

ABSTRACT

Open OnDemand supports Interactive HPC web applications enabling the interactive and distributed environments for Jupyter and RStudio running on an HPC cluster. These web applications provide a simple user-interface for building and submitting the batch job responsible for launching the interactive environment as well as proxying the connection between the user's browser and the web server running on the cluster. Support for distributive computing through a Jupyter notebook and RStudio session is provided by an Apache Spark cluster launched concurrently in standalone mode on the allocated nodes within the batch job. Alternatively, users can directly use the corresponding MPI bindings for either R or Python.

This paper describes the design of Interactive HPC web applications on an Open OnDemand deployment for launching and connecting to Jupyter notebooks and RStudio sessions as well as the architecture and software required for supporting Jupyter, RStudio, and Apache Spark on the corresponding HPC cluster. Singularity can be leveraged for packaging and portability of this architecture across HPC clusters. This paper also discusses the challenges encountered in providing interactive access to HPC resources that are in need of general solutions.

CCS CONCEPTS

• **Software and its engineering** → **Client-server architectures**; • **Information systems** → *Web interfaces*; • **Human-centered computing** → *Web-based interaction*; • **Security and privacy** → *Access control*; *Distributed systems security*; • **Computer systems organization** → *Client-server architectures*;

KEYWORDS

Open OnDemand, High Performance Computing, Interactive, Web platform, Jupyter, RStudio, Spark, OSC

ACM Reference Format:
Jeremy W. Nicklas, Doug Johnson, Shameema Oottikkal, Eric Franz, Brian McMichael, Alan Chalker, and David E. Hudak. 2018. Supporting distributed, interactive Jupyter and RStudio in a scheduled HPC environment with Spark using Open OnDemand. In *PEARC '18: Practice and Experience in Advanced Research Computing, July 22–26, 2018, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3219104.3219149

PEARC '18, July 22–26, 2018, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-6446-1/18/07…$15.00
https://doi.org/10.1145/3219104.3219149

1 INTRODUCTION

Researchers want to apply large-scale computation to new disciplines and with new tools. Meeting the needs of these communities require features not readily supported on today's HPC clusters, like web user interfaces and interactive access to applications. However, HPC clusters are a cornerstone of large-capacity research computing and leveraging these existing investments is cost-effective relative to replicating dedicated environments. Open OnDemand is an open source software project providing web, graphical and interactive access for HPC clusters [5] that is based on the Ohio Supercomputer Center's (OSC) original OnDemand portal [4]. More than just an "out-of-the-box" solution for existing cluster services, the authors envision OnDemand as a platform that can be developed upon to provide new capabilities on existing HPC clusters. In this paper, we describe interactive applications in OnDemand for big data processing with Apache Spark [21] accessed via either Jupyter [6] or RStudio [14].

One of the goals of the Open OnDemand project is to make it easier for HPC users to access HPC resources with the intention to lower the barrier of entry into HPC computing for new users as well as improve the productivity of current HPC users. A user will only need a locally installed browser to access the OnDemand portal

and its provided web applications. Users are not required to install any other third-party software beyond this. The OnDemand portal provides a web interface for file management, job management and monitoring, launching and connecting to interactive jobs, as well as command line shell access for everything else. One of the benefits of OnDemand over a traditional web service is that all the web applications are running as the user. This allows for accounting and security by relying on the implicit user space provided by the Linux kernel.

The OnDemand portal comes with a set of core applications built using Ruby and the OnDemand AppKit library as well as Node.js and WebSocket technology. These applications include the Dashboard App, Shell App, Files App, File Editor App, Active Jobs App, and the Job Composer App. A few of these web applications leverage the OnDemand AppKit library which provides a plug-in friendly resource manager adapter. This library extends the capabilities of the web application by providing a generic interface to a multitude of supported resource managers: Torque, Slurm, LSF, and PBS Pro. The Dashboard in particular makes use of the OnDemand AppKit library to create and maintain interactive jobs, a job that launches a web application server on a node in the cluster that the user can connect back to and interact with (e.g., a Jupyter Notebook server) in their browser. A system administrator can create an Interactive App that the Dashboard serves by supplying a set of custom YAML configuration files that describe the app and how it is submitted as well as a template of shell scripts that are run within the job.

Two popular scientific web applications that OSC OnDemand supports is Jupyter Notebook and RStudio Server. The Jupyter Notebook is an interactive web based notebook that supports Python, among many other languages. It has a rich input interface that integrates code development, documentation, and visualization to implement a Read-Eval-Print Loop (REPL) interface. Similarly, RStudio Server provides an integrated development environment web application with support for running and debugging programs written in R. Both of these have been ported and installed on OSC OnDemand as Interactive Apps since April 2017. Since then the Jupyter Notebook App has been launched 782 times from 182 different OSC users and the RStudio Server App has been launched 1,337 times from 162 users. Both of these Interactive Apps are also hosted on OnDemand installations at 5 different institutions. OSC OnDemand recently added a Jupyter with Spark Interactive App in November 2017 that has since been launched 97 times by 41 different OSC users. The Jupyter with Spark Interactive App was also featured in a workshop hosted at the University of Cincinnati in March 2018 where all the attendees were able to simultaneously launch and work with their own Jupyter and Spark instances on the OSC cluster in their browsers.

Figure 1 shows an example of the Jupyter with Spark Interactive App hosted on OSC's OnDemand portal. The user fills in the configurable requisite information for the Interactive App: project to be charged, number of hours to run job, number of requested nodes, etc. The Interactive App then submits a job that launches a Spark cluster and Jupyter Notebook server. The user is able to connect to the Jupyter Notebook server in their browser when the job starts. The Dashboard is responsible for submitting and managing the Interactive App's job as well as to generate the HTML link

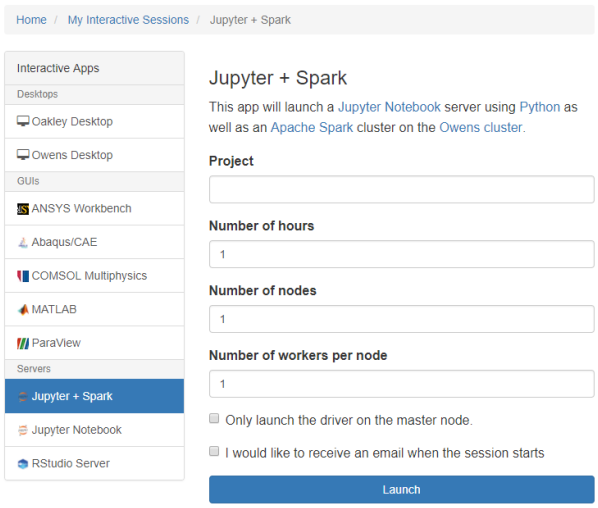

Figure 1: A screenshot of the Jupyter + Spark Interactive App hosted on OSC OnDemand

that the user clicks on to establish the connection back to the web application server running in the job. A key feature of the OnDemand portal is that it provides a reverse proxy to securely bridge the connection between the client browser and the applications running on the HPC compute nodes. This eliminates the need for an HPC user to setup an SSH tunnel to establish this connection.

In a naive deployment both the Jupyter and RStudio web application servers are setup by default to listen on a high numbered network port on the loopback interface. The application server runs as the user that started the server with no authentication enabled, and the user connects locally from their web browser. While local deployment of the application server is simple, integration in a multi-user HPC environment is difficult. Both Jupyter and RStudio support authentication and execution as the authenticated user, but integration with HPC resource managers do not exist out of the box. Furthermore, extending these application servers for distributed execution presents additional challenges. The languages supported by these application servers support a variety of parallel execution frameworks such as the message passing interface (MPI) and Spark, but supporting these further complicates the integration of the application servers into an HPC environment.

It should also be noted that although VNC is not necessarily a web application server and browser client, it can leverage websockify [12] for its web application server and noVNC [11] for its browser-based VNC client. The websockify tool translates the front-facing WebSocket traffic to the normal socket traffic that the VNC server expects. This extends Interactive App support to include X11-client applications that can be normally run within a VNC session. OSC's OnDemand portal currently offers a variety of Linux desktops as well as ANSYS Workbench, Abaqus, COMSOL, MATLAB, and ParaView.

In the following sections we will describe how Open OnDemand supports secure end-to-end communication from a client web browser to serial and parallel HPC jobs executing a Jupyter

Notebook or RStudio Server with a dedicated Spark cluster running in Standalone mode.

2 RELATED WORK

A number of HPC centers have implemented web-based interactive computing environments for their HPC resources. Texas Advanced Computing Center's (TACC) visualization portal includes a web-based VNC application, providing enhanced security and portability and enabling users to start a desktop session from any browser [17]. TACC's visualization portal also offers interactive, web-based Jupyter Notebook and RStudio integration. Advanced Research Computing Technology Service of University of Michigan also provides Jupyter and RStudio integration through a web portal [8]. Minnesota Supercomputing Institute offers an interactive computing environment for Jupyter through a web portal as well [10]. Microsoft's Azure HD insight offers Spark integrated with Jupyter and R server to interactively analyse Big Data on a web platform [9]. The Amazon EMR data processing cluster allows users to create a JupyterHub interface to Spark to process and visualize data on the web [22]. It also offers an RStudio interface to Spark [23]. Google's Cloud Dataproc allows users to integrate the Spark engine with Jupyter and RStudio [13].

3 INTERACTIVE HPC WEB SERVERS

A few of the problems an administrator faces when deploying a web application server are listed as follows:

- restricting access of the application server to only verified and allowed users
- maintaining secure communication between the user's browser and the application server

These are typically handled by launching the application server on a security hardened machine listening on the loopback interface to a high numbered port (> 1024). A reverse proxy server is then exposed to the public network that handles the HTTP over SSL traffic and proxies it back to the local application server. Authentication can also be handled by the reverse proxy server or at the application server layer.

By starting the application server on the loopback interface it limits all incoming connections to within the local machine. A security hardened machine will have a much smaller surface of vulnerability leaving external access to the application server only through the reverse proxy server. This not only allows for the authentication layer to be placed at either the reverse proxy or application server, but also allows for the encrypted SSL layer to be handled by the reverse proxy. This is beneficial as setting up SSL certificates for a web application can be difficult if at all possible.

If a user at an HPC center wishes to start up a web application server within a batch job or from a login node they will lose the benefits of a security hardened machine and the reverse proxy server acting as the gatekeeper. Historically, the secure connection to the application server is established by setting up an SSH tunnel through the login node from the user's local machine to the application server running within the HPC center's internal network. The user is then expected to navigate in their browser to the port on their local machine they used to establish the SSH tunnel with. This can lead to issues where HTTP redirects and cookie domains point to the host and port of the machine the application server is running on rather than the client's localhost and port that the SSH tunnel is listening on. This also does not resolve the issue with restricting access to just the owner of the application server.

For the case of starting a web application server on the login node the benefits of listening on the loopback interface are nonexistent as all users on that node will be able to connect to the application server. Starting a web application server on the loopback interface from within a batch job on a compute node requires that the compute node be security hardened and allow only SSH access for just the owner of that job. Even with this restriction there are still many possible security concerns with this model as previous user processes may not have been properly cleaned or some other service running on the node allows user access. In short, the application server will need to provide some form of authentication and session management to restrict access to just the owner. We will discuss the authentication layers provided within Jupyter Notebook in section 3.2 and RStudio Server in section 3.3.

3.1 OnDemand Reverse Proxy

The OnDemand portal comes with a dynamic reverse proxy that is disabled by default but can be enabled by a system administrator. Once enabled the OnDemand portal can proxy the secure HTTP over SSL traffic from the user's browser to any configured web resource within the HPC center's internal network. The reverse proxy takes advantage of the authentication layer provided by the OnDemand portal to restrict access to only verified HPC users. After a user successfully authenticates with OnDemand he or she can then utilize the reverse proxy to communicate with an application server listening within the internal network. This is functionally equivalent to the SSH tunnel discussed previously.

The reverse proxy packaged in OnDemand is also responsible for rewriting the URLs that may appear in redirect and cookie headers. This is essential in any reverse proxy to avoid bypassing the reverse proxy. This is an issue when using an SSH tunnel to securely proxy HTTP traffic inside an HPC center's internal network.

A drawback of the OnDemand reverse proxy is that the information about the host and port to proxy the request to needs to be included in each HTTP request sent to the OnDemand portal. This is due to the stateless nature of web services and in particular the OnDemand dynamic reverse proxy. This is not an issue for SSH tunnels as the host and port is originally supplied when establishing the SSH tunnel and remains static thereafter.

The solution taken by the OnDemand portal is to require the host and port to be specified within the path of the URL request. The URL specification for the OnDemand dynamic reverse proxy is given as:

```
https://example.com[/handler][/host][/port][/path]
```

where `handler` defines which reverse proxy handler to use, `host` and `port` define the application server to proxy the request to, and `path` is the URL path consumed by the application server. The OnDemand dynamic reverse proxy accepts two types of reverse proxy handlers `/node` and `/rnode` that describe how the URL path is constructed and consumed by the application server.

The first reverse proxy handler is accessed with the URL path segment `/node`. This instructs the reverse proxy to construct a

URL path that is identical to the original URL path when sending the request to the application server running on the back-end. For example, when navigating to:

`https://example.com/node/node01/8080/index.hml`

the OnDemand reverse proxy constructs and makes the following URL request:

`http://node01:8080/node/node01/8080/index.html`

to access the application server. Note that the URL paths are identical in both requests. This is used by application servers that can be configured under a sub-URI, e.g., Jupyter Notebook server.

The second reverse proxy handler is accessed with the URL path segment `/rnode`. This instructs the reverse proxy to construct a URL path that is stripped of the preceding reverse proxy information when sending the request to the application server running on the back-end. For example, when navigating to:

`https://example.com/rnode/node01/8080/index.html`

the OnDemand reverse proxy constructs and makes the following URL request:

`http://node01:8080/index.hml`

to access the application server. Note that the preceding component of the original URL path consumed by the reverse proxy (`/rnode/node01/8080`) is stripped away in the request consumed by the application server. This is used by application servers that use *relative* URL links and not absolute file paths when linking to pages or pointing to assets and images, e.g., RStudio Server.

Figure 2 outlines the steps that occur right after the web application server starts within the batch job (on a compute node) that was previously submitted by the Dashboard App up until the user establishes a connection to the application server through the OnDemand reverse proxy. The connection information containing the host and port is output to a file on a shared file system immediately after the application server is successfully started. Behind the scenes the Dashboard App reads this connection information and generates an HTML link when the user browses to the Dashboard. This link contains the URL path with the reverse proxy handler that the web application server supports as well as the corresponding host and port to proxy the request to. Finally, the user's browser is redirected to the OnDemand dynamic reverse proxy when the user clicks the link in the Dashboard and establishes a connection with the web application server running on the compute node. The Dashboard only facilitates this connection and is not necessary after the connection has been established.

By incorporating the reverse proxy information into the URL path OnDemand is constrained by the variety of possible web application servers it can support. In our experience though this hasn't been a limiting factor with any of the web application servers we have currently worked with: Jupyter Notebook server, RStudio Server, COMSOL Server, Shiny Server, and noVNC+websockify.

The OnDemand reverse proxy much like an SSH tunnel only establishes the connection between the client and the web application server running on the internal HPC network. It does not prevent other HPC users from also connecting to your web application server. Therefore, a developer of an Interactive App should be fully aware that the web application server started in the job will need its own authentication layer to restrict access to the process owner.

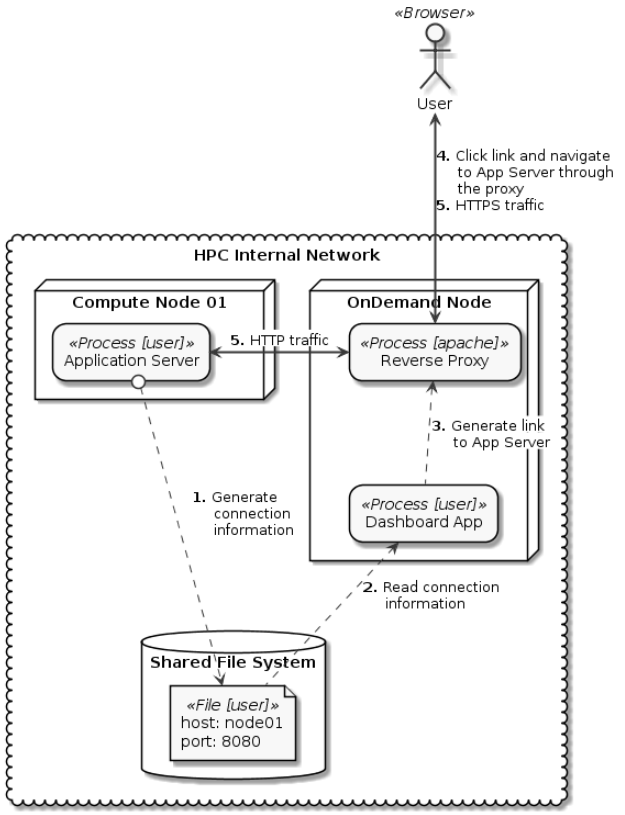

Figure 2: A diagram outlining the steps taken from when a user's application server launches in a batch job until the user establishes a connection with the application server through the OnDemand reverse proxy.

3.2 Jupyter Notebook

The Jupyter Notebook server uses a two-process kernel architecture that decouples the Notebook server process from the kernel process that performs the evaluation in the REPL environment. This allows us to decouple the Jupyter Notebook installation from the various Python installations on the HPC cluster. Therefore, updating the Jupyter Notebook software and its dependencies won't break or alter the highly-optimized scientific Python libraries managed by the HPC Scientific Applications team.

A separate installation of Python 3 and Jupyter Notebook can exist in an isolated directory such as `/usr/local/python3-notebook`. A Jupyter Notebook server is launched from within a batch job using a custom dynamically generated configuration file. The Notebook configuration file at a minimum needs to specify the port to listen on, the randomly generated hashed password used for authentication, the sub-URI that the OnDemand reverse proxy will use, and to disable the browser that is by default launched when a Notebook server is started. Further configuration options can be supplied to simplify the connection process between the user and the Notebook server.

The user will then be able to connect to the Jupyter Notebook server by visiting the following OnDemand URL:

```
https://example.com/node[/host][/port]/
```

where `host` is the host of the first node allocated to the batch job and `port` is the port the Jupyter Notebook server is listening on. The user will then be prompted for the password specified in the Notebook configuration file keeping the server secure from other HPC users. Upon successful authentication the user will then be able to create new notebooks from a list of available kernels.

Kernels can be generated for each of the available Python modules hosted by the HPC cluster as well as for various other languages that support iPython kernels (e.g., Julia). The Jupyter Notebook kernel spec is serialized in a JSON format that provides a kernel name, the language of the kernel, a list of command line arguments used to start the kernel, and a dictionary of environment variables set for the kernel. For each of the modules installed on the HPC cluster a custom kernel spec is generated that points to a corresponding wrapper script. The wrapper script is responsible for loading the given HPC cluster module environment before launching the system-installed kernel executable. A user can now leverage the optimized Python scientific libraries installed on the HPC cluster from within their Jupyter Notebook server.

3.3 RStudio Server

RStudio Server decouples the web server component (`rserver` process) from the individual R sessions (`rsession` processes) by maintaining them as separate processes. The `rserver` process is responsible for serving the login page and starting the `rsession` process. It is also responsible for routing the traffic from the web browser to the running session. These two processes communicate over a Unix domain socket using JSON messages. The `rsession` process is responsible for loading R as a library and making calls to this library when needed to evaluate R expressions.

RStudio Server is typically run as a privileged user. When a user accesses RStudio Server in their browser they will be prompted for their local username and password. RStudio Server is integrated with PAM for user authentication by default and after a user successfully authenticates it will start an `rsession` process as the authenticated user. All further web traffic from the authenticated user will be proxied to the user's `rsession` process. This workflow is difficult to port to a per-user server workflow running on an HPC cluster.

The first complication is that RStudio Server will write its state to the common location `/tmp/rstudio-server`. This introduces file ownership and permission issues when multiple users start `rserver` processes. Even if an HPC cluster only allows a single user on a compute node, it is not guaranteed the previous user properly cleaned up these state files. There are two solutions currently employed to solve this issue. The first is using the PRoot tool [19] which provides a user-space implementation of `mount --bind` by relying on `ptrace`. A fake bind mount can be created for the `rserver` process by launching it with:

```
proot -b "$(mktemp -d):/tmp" rserver [OPTIONS]
```

Using PRoot may incur a performance penalty during I/O operations. An alternative solution is to leverage containers, in particular Singularity [7], to isolate the `/tmp` directory from the host file system. A Singularity image can be provided with both R and RStudio Server installed under it. To keep the image portable the HPC cluster's R packages would be installed and maintained on the host file system and a bind mount would be performed to include them within the Singularity container. This would allow for the HPC cluster's optimized scientific R libraries to be decoupled from the Singularity image.

Another complication is the lack of privileges a user has to use PAM authentication. This is resolved by supplying a custom authentication script that is called by the `rserver` process during the authentication operation. RStudio Server allows you to supply a PAM helper script that can be used for authenticating a user. This script is called with the username as the first and only argument and the password piped from standard input. RStudio Server treats it as a successful authentication if the script returns a zero exit status and a failed authentication otherwise. It is trivial to write a shell script that compares a user supplied password to a randomly generated password written to a secure file or saved as an environment variable.

RStudio Server is meant to be run as a privileged system user that starts an `rsession` process for each authenticated user. In doing so it sanitizes the environment for security purposes when forking the `rsession` process. This is problematic for the per-user RStudio Server model as we'd like to keep the same environment in the `rsession` process as was set when launching `rserver` as the user. This is especially so for the case when we integrate Spark with RStudio discussed in section 4.2. This can be resolved with a wrapper script around the `rsession` executable that re-invokes the same environment used to launch `rserver`. The custom wrapper script is provided as a command line argument when launching the `rserver` executable.

The user is then able to connect to the RStudio Server by visiting the following OnDemand URL:

```
https://example.com/rnode[/host][/port]/
```

where `host` is the host of the first node allocated to the batch job and `port` is the port the RStudio Server is listening on. The user will then be prompted to enter their username and the password that was generated in the batch script when starting up the RStudio Server. Upon successful authentication the user will then be presented with a web-based IDE and an R REPL workspace in their browser.

4 PARALLELIZE INTERACTIVE HPC APPS WITH SPARK

A Spark cluster is most easily launched in Standalone cluster mode within a batch job running as a user. This has the fewest moving parts and doesn't require privilege escalation. Deploying Spark in Standalone mode consists of a single Standalone Master process that manages the resources for the Spark Standalone cluster. It also consists of one or more Standalone Worker processes that are connected back to the Master. Standalone mode is just one of the many cluster modes that Spark supports.

Irrespective of the Spark deployment mode, a standalone application (e.g., the Python Notebook) will construct a `SparkContext` that becomes the driver program that connects to the Spark resource manager and requests resources. The resource manager launches executors on the workers that then register themselves back with the driver program. Depending on the actions and transformations

taken in the user application, tasks are then sent to the executors where they are evaluated and returned to the driver program. When the driver program exits all executors will be terminated and the cluster resources will be released.

A few of the issues encountered with launching Spark in Standalone mode from within a batch job include: starting the Worker processes so that they are managed by the HPC cluster resource manager, securing the Spark communication protocols, and securing the Spark Web UI.

The Spark Standalone cluster mode launch scripts included in the Spark installation are not sufficient to launch the Master and Worker processes within a batch job on an HPC cluster. Instead the Master process is explicitly started within the batch script and the script is blocked until the Master process is running and listening on the desired port. A wrapper script to launch the Worker processes is then dynamically generated with the connection information from the Master process and the commands to load them in an identical software environment. The HPC cluster resource manager's underlying tool (e.g., `pbsdsh` for PBS and `srun` for Slurm) is then used to distribute and call the Worker wrapper script on the allocated nodes. This allows the Worker processes to be properly managed by the HPC cluster's resource manager.

Spark can be configured to secure the communication protocol via a shared secret. This is randomly generated early on in the batch job and is written to a secure Spark properties file. The path to this Spark properties file is passed as a command line argument when starting the Master process, Worker process, and constructing the `SparkContext` within the user application. The Spark Web UI can also be secured through the use of `javax.servlet` filters.

After a Spark Standalone cluster is started, the given user application is then started and a `SparkContext` with details about the Master process is constructed within the application. Both Python and R support constructing a `SparkContext` and connecting to a Spark cluster. We describe how to do this for both the Jupyter Notebook server in section 4.1 and RStudio Server in section 4.2.

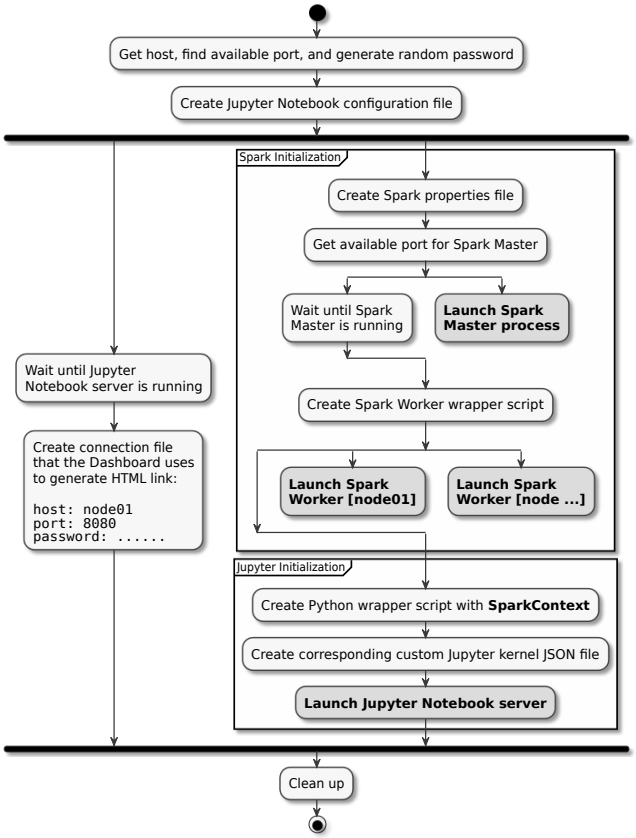

Figure 3: An activity diagram for a job script generated and submitted by the Dashboard App that launches a Spark cluster in Standalone mode on a set of nodes as well as a Jupyter Notebook server that the user can connect to through the OnDemand reverse proxy.

4.1 Jupyter with Spark

After the Spark cluster is started a Jupyter Notebook server is configured and launched in a very similar fashion to the discussion in section 3.2. The difference being that the Spark state is now encapsulated within the Python wrapper script that is launched by the custom Jupyter Notebook kernel. Previously we used these custom kernels and corresponding wrapper scripts to load the software environment provided by the HPC cluster before launching the kernel executable. Now a custom kernel and wrapper script will be generated that also includes the required `SparkContext` information as well as the path to a Python startup script provided by the Spark installation. Figure 3 outlines the workflow of a batch job that launches both a Spark Standalone cluster and Jupyter Notebook server that connects to it.

The Python wrapper script needs to load the same software environment that the Spark Worker processes are running under so that the driver uses the same same software package and libraries as the executors. The environment variables `PYTHONPATH` and `PYTHONSTARTUP` are defined to point to the necessary Python libraries and startup script respectively that are provided under the Spark installation location. The `pyspark-shell` process and its corresponding command line arguments, which must include the Spark properties file that contains the shared secret, are set in the `PYSPARK_SUBMIT_ARGS` environment variable. This environment variable contains all the necessary connection information used to setup a `SparkContext` to our Spark cluster running within the batch job.

When a user accesses the Jupyter Notebook server they will be able to select the custom kernel that launches this Python wrapper script when starting a new Notebook. From within the Notebook running this kernel they can immediately start submitting tasks to the `SparkContext` through the global variable `sc`. The Standalone cluster mode in Spark currently only supports a simple FIFO scheduler across applications and by default the first Notebook will acquire all cores in the cluster. This limits you to only being able to run one Notebook with a `SparkContext` at a time. After you close this Notebook the Spark cluster resources will be released and you can open another Notebook with a `SparkContext`.

4.2 RStudio with Spark

There are two competing R packages for connecting to a running Spark cluster: `sparkR` [18] and `sparklyr` [16]. The `sparkR` package is supported by the Apache Spark software community whereas the `sparklyr` package is supported by the RStudio development community. This paper focuses on incorporating the latter into the RStudio Server running as the user within a batch job.

As with the Jupyter Notebook server, the steps necessary when configuring and starting the RStudio Server `rserver` process are nearly identical to those discussed in section 3.3. The difference being that the required `SparkContext` information will now be encapsulated in the `rsession` wrapper script and a custom startup profile script that is responsible for creating the `SparkContext` object is generated. A couple disadvantages of the `sparklyr` package when compared with `sparkR` is its lack of support with defining the `SparkContext` configuration through environment variables as well as its lack of a startup profile file that builds the `SparkContext` when the `rsession` starts. The `sparklyr` package intends for the user to define the `SparkContext` within the RStudio IDE itself. This can be burdensome for a user whose Spark cluster running in Standalone mode is automatically launched within the batch job using a randomly generated shared secret and port.

The `rsession` wrapper script can supply a subset of the Spark shell command line arguments through the `sparklyr.shell.args` environment variable, in particular the Spark properties file that contains the shared secret. This wrapper script also sets the `R_PROFILE_USER` environment variable to the dynamically generated R startup profile file that is modeled after the file provided by the `sparkR` package. This profile script is responsible for resetting the `R_PROFILE_USER` environment variable to another generic startup profile, adding `sparklyr` as a default loaded package, and creating a `SparkContext` for the Spark cluster in a global variable named `sc`. The `R_PROFILE_USER` needs to be reset as it will be called whenever another R process is forked, e.g., during package installation. It is not necessary to create a `SparkContext` during package installations.

When a user accesses the RStudio Server for the first time and is provided an `rsession` process they will be able to immediately begin submitting tasks to the `SparkContext` through the global variable `sc`. The user will only be provided a single `rsession` process by the `rserver` process. When the user closes the `rsession` the Spark cluster resources are released.

5 CONCLUSIONS

In this paper, we describe interactive applications in OnDemand for big data processing with Apache Spark accessed via either Jupyter or RStudio. This solution leverages features provided by the OnDemand platform for web applications interacting with cluster schedulers and nodes. The majority of the work for this project is the installation and configuration of the three applications (Spark, Jupyter and RStudio) on the cluster. This shows that OnDemand can be extended to provide new capabilities on existing HPC clusters, giving new capabilities to researchers in a cost-effective manner.

6 FUTURE WORK AND CHALLENGES

Distributed support for Jupyter and RStudio through Open OnDemand are currently limited to the Spark framework. Supporting MPI bindings for Python using mpi4py [1–3] or Rmpi [20] for R is a logical extension of our work. Challenges for supporting MPI include ensuring compatibility between the MPI bindings and the implementations of MPI available on the system, integration with the supported MPI job launcher, and a more complete understanding of the how processes must be launched along with the interactive interfaces.

Other future extensions to the work include: allowing users to execute a notebook that was created in an interactive session in a non-interactive batch job; support for Shiny apps [15]; giving users more control over the launch options for the Spark cluster; support for shared Spark clusters that are not running as part of the user's batch job.

A general challenge exists for supporting interactive environments at HPC centers: scheduling resources for interactive jobs when the end user is available. This requires either keeping some fraction of the hardware accessible for immediate scheduling for the interactive workload, the ability for end users to request access to resources via an advance reservation mechanism so the resources are available when they need to perform the interactive work, or over-subscription of resources for interactive jobs.

ACKNOWLEDGMENTS

This work is supported by the National Science Foundation of the United States under the award NSF SI2-SSE-1534949.

REFERENCES

[1] Lisandro Dalcín, Rodrigo Paz, and Mario Storti. 2005. MPI for Python. *J. Parallel and Distrib. Comput.* 65, 9 (2005), 1108 – 1115. https://doi.org/10.1016/j.jpdc.2005.03.010

[2] Lisandro Dalcín, Rodrigo Paz, Mario Storti, and Jorge D'Elía. 2008. MPI for Python: Performance improvements and MPI-2 extensions. *J. Parallel and Distrib. Comput.* 68, 5 (2008), 655 – 662. https://doi.org/10.1016/j.jpdc.2007.09.005

[3] Lisandro D. Dalcín, Rodrigo R. Paz, Pablo A. Kler, and Alejandro Cosimo. 2011. Parallel distributed computing using Python. *Advances in Water Resources* 34, 9 (2011), 1124 – 1139. https://doi.org/10.1016/j.advwatres.2011.04.013 New Computational Methods and Software Tools.

[4] David E. Hudak, Thomas Bitterman, Patricia Carey, Douglas Johnson, Eric Franz, Shaun Brady, and Piyush Diwan. 2013. OSC OnDemand: A Web Platform Integrating Access to HPC Systems, Web and VNC Applications. In *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery (XSEDE '13)*. ACM, New York, NY, USA, Article 49, 6 pages. https://doi.org/10.1145/2484762.2484780

[5] David E. Hudak, Douglas Johnson, Jeremy Nicklas, Eric Franz, Brian McMichael, and Basil Gohar. 2016. Open OnDemand: Transforming Computational Science Through Omnidisciplinary Software Cyberinfrastructure. In *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale (XSEDE16)*. ACM, New York, NY, USA, 43:1–43:7. https://doi.org/10.1145/2949550.2949644

[6] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathon Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, Carol Willing, and Jupyter Development Team. 2016. Jupyter Notebooks - a publishing format for reproducible computational workflows. , 87–90 pages. https://doi.org/10.3233/978-1-61499-649-1-87

[7] Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. 2017. Singularity: Scientific containers for mobility of compute. *PLOS ONE* 12, 5 (05 2017), 1–20. https://doi.org/10.1371/journal.pone.0177459

[8] Dan Meisler. 2016. New ARC Connect service provides desktop graphical interface for HPC resources. http://arc-ts.umich.edu/new-arc-connect-service-provides-desktop-graphical-interface-for-hpc-resources/

[9] Microsoft. [n. d.]. Apache Spark for Azure HDInsight. https://azure.microsoft.com/en-us/services/hdinsight/apache-spark/

[10] Michael Milligan. 2017. Interactive HPC Gateways with Jupyter and Jupyterhub. In *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact (PEARC17)*. ACM, New York, NY, USA, 63:1–63:4. https://doi.org/10.1145/3093338.3104159

[11] noVNC Team. 2018. noVNC. https://github.com/novnc/noVNC

[12] noVNC Team. 2018. websockify. https://github.com/novnc/websockify
[13] Reza Rokni and James Malone. 2017. Google Cloud Platform for data scientists: using Jupyter Notebooks with Apache Spark on Google Cloud. https://cloud.google.com/blog/big-data/2017/02/google-cloud-platform-for-data-scientists-using-jupyter-notebooks-with-apache-spark-on-google-cloud
[14] RStudio Team. 2015. *RStudio: Integrated Development Environment for R.* RStudio, Inc., Boston, MA. https://www.rstudio.com/
[15] RStudio Team. 2018. Shiny from RStudio. https://shiny.rstudio.com/
[16] RStudio Team. 2018. sparklyr: R interface for Apache Spark. https://spark.rstudio.com/
[17] TACC Team. 2018. TACC Visualization Portal. https://vis.tacc.utexas.edu/
[18] Shivaram Venkataraman, Zongheng Yang, Davies Liu, Eric Liang, Hossein Falaki, Xiangrui Meng, Reynold Xin, Ali Ghodsi, Michael Franklin, Ion Stoica, and Matei Zaharia. 2016. SparkR: Scaling R Programs with Spark. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16)*. ACM, New York, NY, USA, 1099–1104. https://doi.org/10.1145/2882903.2903740
[19] Cédric Vincent and Yves Janin. 2011. PRoot: a Step Forward for QEMU User-Mode. In *1st International QEMU Users' Forum*. 41–44.
[20] Hao Yu. 2002. Rmpi: Parallel Statistical Computing in R. *R News* 2, 2 (2002), 10–14. http://cran.r-project.org/doc/Rnews/Rnews_2002-2.pdf
[21] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud'10)*. USENIX Association, Berkeley, CA, USA, 10–10. http://dl.acm.org/citation.cfm?id=1863103.1863113
[22] Tom Zeng. 2016. Run Jupyter Notebook and JupyterHub on Amazon EMR. https://aws.amazon.com/blogs/big-data/running-jupyter-notebook-and-jupyterhub-on-amazon-emr/
[23] Tom Zeng. 2016. Running sparklyr - RStudio's R Interface to Spark on Amazon EMR. https://aws.amazon.com/blogs/big-data/running-sparklyr-rstudios-r-interface-to-spark-on-amazon-emr/