Remote Visualization With ParaView

Cory Quammen
Kitware, Inc.

Kitware

Overview

Part 1 - Getting connected

- ParaView client/server architecture
- Running and connecting the client and server
- Connection types
- Connecting to remote clusters

Part 2 - Remote visualization

- What happens where
- Remote vs. local rendering
- Speeding up remote rendering

Kitware

Part 1
Getting connected

ParaView Architecture

ParaView client - application with user interface

- Runs on the desktop to visualize local data

ParaView server (pvserver) - server application

- Runs on remote system
- Can be built with MPI for parallel visualization on clusters

Kitware

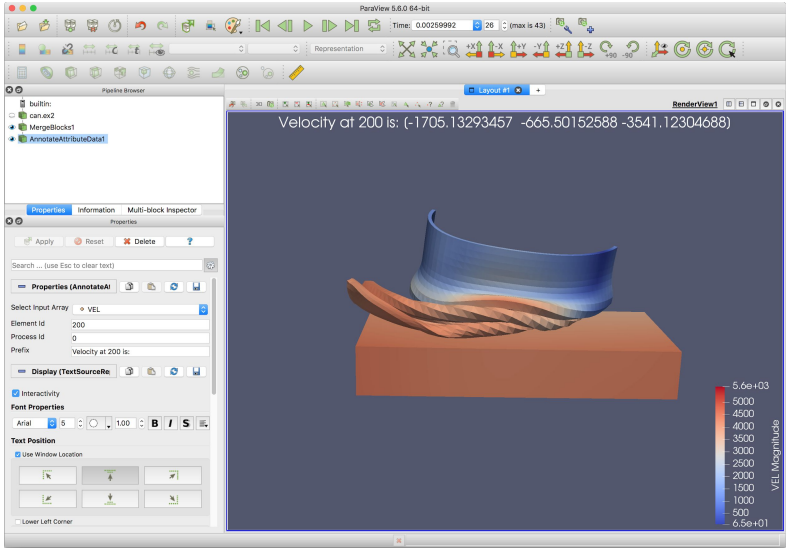
ParaView client application

Is a GUI application

Scriptable with Python

Connects to a server for visualization

By default, is connected to a built-in server in the same process, offering standalone visualization capabilities.

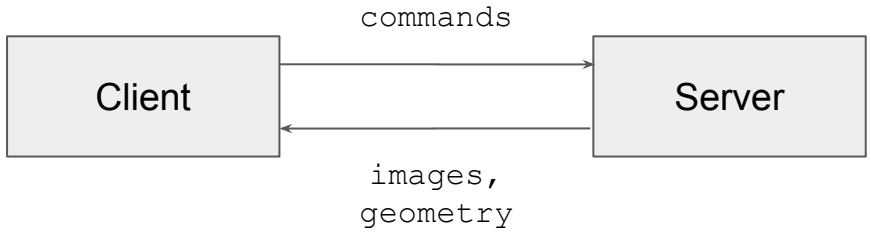


Kitware

ParaView server

Executable named `pvserver` that runs on a remote system where data resides

- No GUI available (do not forward X11 to your client workstation)
- Connects to a client that drives the visualization session



Kitware

ParaView server - graphics requirements

Can use graphics acceleration if available

- Can use X11 to access graphics accelerators
- If NVIDIA nodes are available, building with EGL is possible (no X11 required)
- If neither X11 nor EGL are available, can do software rendering with offscreen Mesa OpenGL library

When running `pvserver` remotely through SSH, do not enable X11 forwarding with `-x` or `-y`

- This could cause rendering to happen on the client!

Kitware

ParaView server - MPI

`pvserver` can be built with or without MPI (with is recommended)

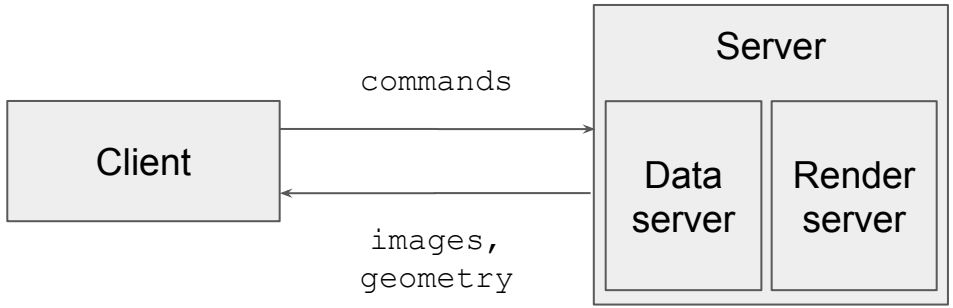
- In both cases, connecting the client to the server is the same
- Client connects to rank 0 on the server, process 0 coordinates ranks

Application binaries provided by Kitware are built with MPICH 3.2

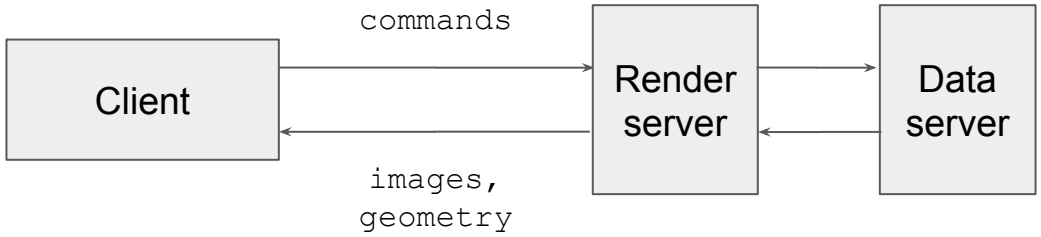
- This is probably not optimal on your system
- If system has hi-speed interconnect, you will want to build `pvserver` with MPI for the interconnect

Side note - data server, render server

`pvserver` is actually two servers, a data server and a render server



The two servers can be run separately, but not often used this way



Kitware

Running pvserver

Run `pvserver[.exe]` at a terminal

Linux -

`pvserver` is in same bin/ directory as paraview

macOS -

`pvserver` is in /Applications/ParaView-5.6.0.app/Contents/bin

Windows -

`pvserver.exe` is in C:\Program Files\ParaView-5.6.0*\bin

Kitware

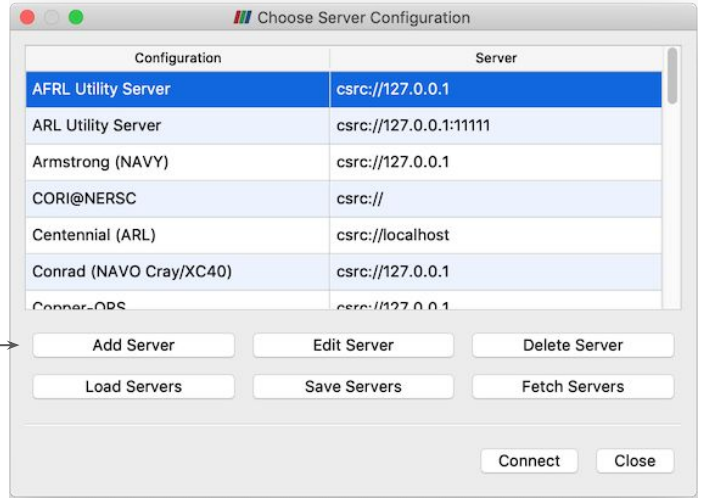
`pvserver` waiting for a connection

```
> pvserver
Waiting for client…
Connection URL: cs://terminus.local:11111
Accepting connection(s): terminus.local:11111
```

Kitware

Connecting the client

In client, clicking the  button brings up connection dialog.

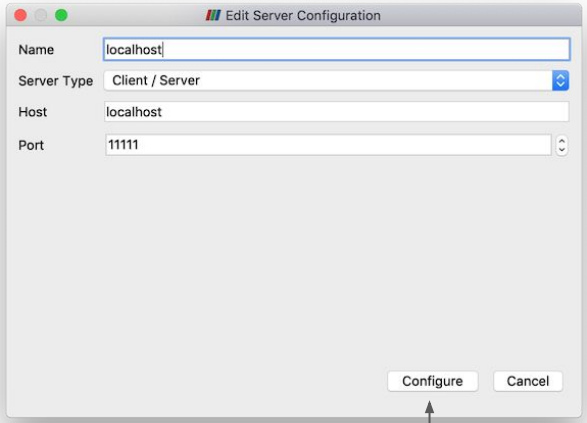


Click here to add a new server

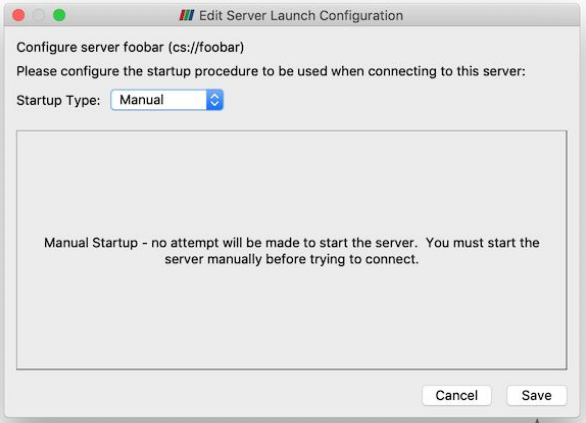
Kitware

Add forward connection to localhost

Add a configuration to connect to a server running on same machine as client



Enter text, click **Configure**

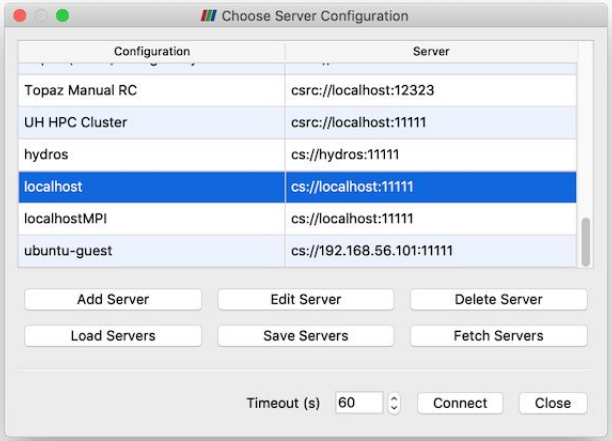


Click **Save**

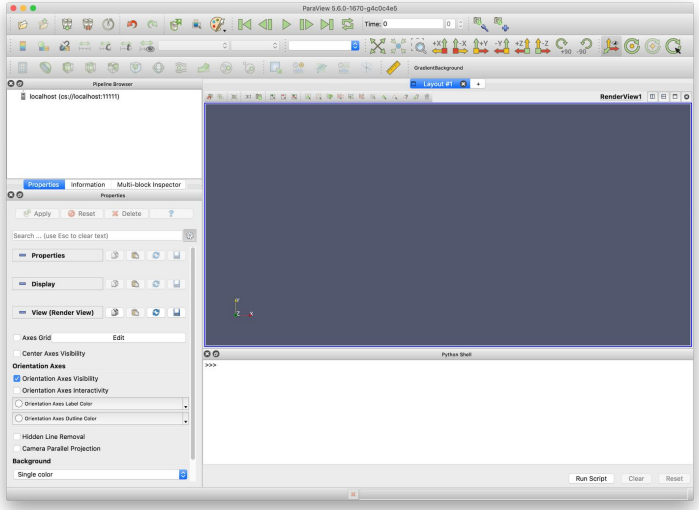
Kitware

Connect to localhost

Select localhost configuration and click **Connect**



Connected server shown here



Kitware

Try it

Navigate to your ParaView installation

Run `pvserver` on your local system with no arguments

Set up a new connection configuration 

Connect to the server from the ParaView client

Disconnect by clicking 

Kitware

Running `pvserver` in parallel

You can run `pvserver` like any other MPI program

```
> mpiexec -np <num processes> pvserver [additional arguments]
```

*Windows version built with MPI requires MS-MPI (free download) to be installed

Connecting to the server is the same as when it is run serially

Running `pvserver` in a job queue

Typically need to submit batch job

```
qsub -A <project to charge time to>
  -N <number of nodes>
  -n <number of cores on each node>
  -q <job to submit job to>
  mpiexec -np <N*n> pvserver
```

Kitware

Forward connections

> `pvserver`

Listens for incoming client connection on port 11111

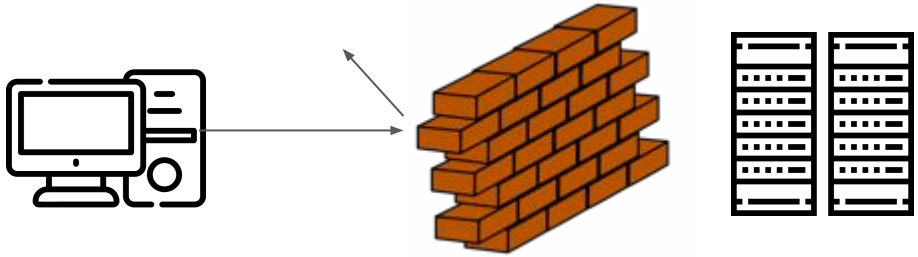
> `pvserver --server-port=22222`

Listens for incoming client connection on custom port 22222

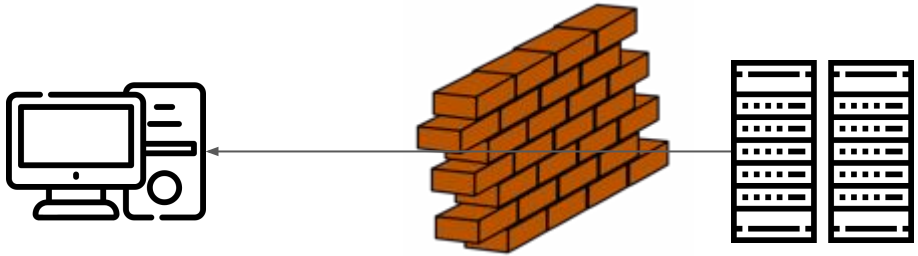
Kitware

If only it were that simple...

No HPC cluster is open to the internet with an open socket connection



Common solution - have the server connect to the client



Kitware

Reverse connections

ParaView client listens for incoming connection, `pvserver` initiates connection to `--client-host` argument

```
> pvserver --reverse-connect --client-host=localhost
```

This is a new type of connection, so we need to add a new connection configuration to the ParaView client

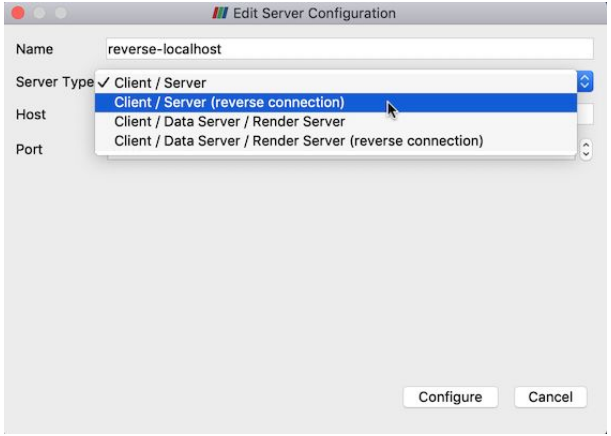
Kitware

Configure reverse connection

Add a server in the **Choose Configuration Server** dialog

Set **Server Type** to “Client / Server (reverse connection)

Click **Configure** and click **Save** on the next screen

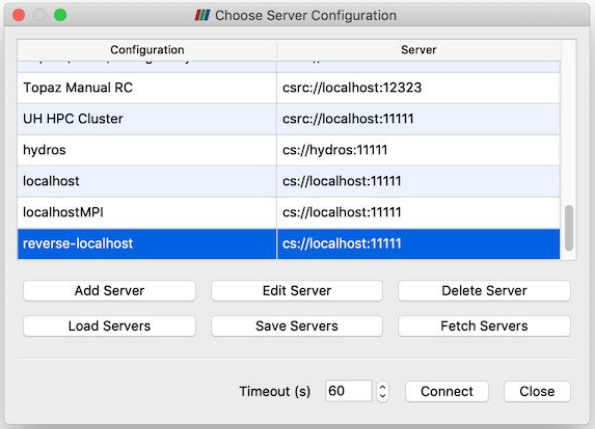


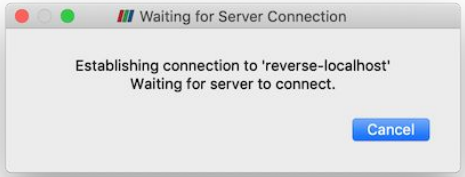
Kitware

Connect the server to the client

Run `pvserver --reverse-connect --client-host=localhost`

Select the **reverse-localhost** configuration and click **Connect**





Server will output:

```
Connecting to client (reverse
connection requested)...
Connection URL: csrc://localhost:11111
Client connected.
```

Kitware

Try it

Set up a new connection configuration 

Name: reverse-localhost

Host: localhost

Port: default (11111)

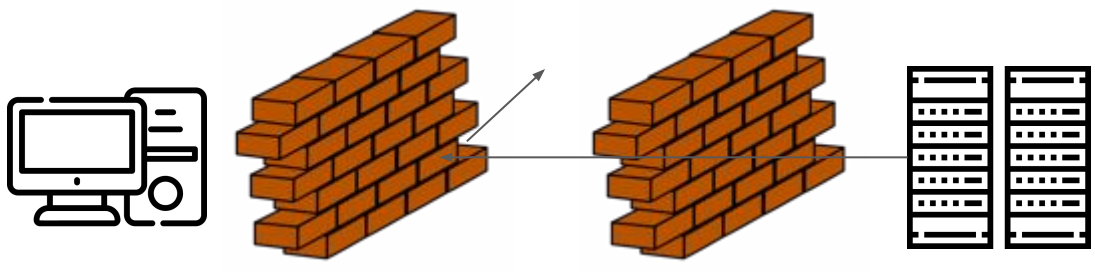
Server Type: “Client / Server (Reverse Connection)”

Connect in the **Choose Server Configuration** dialog

```
> pvserver --reverse-connection --client-host=localhost
```

Kitware

If client is behind a firewall...



A reverse connection alone will not work in this scenario

Outside connections cannot get to ParaView client behind firewall

Kitware

SSH tunneling

Usually systems behind firewalls can be accessed via SSH

SSH allows a mechanism called “tunneling” that can be used to bypass firewall restrictions on network connections in a secure way by sharing the SSH connection with other network connections

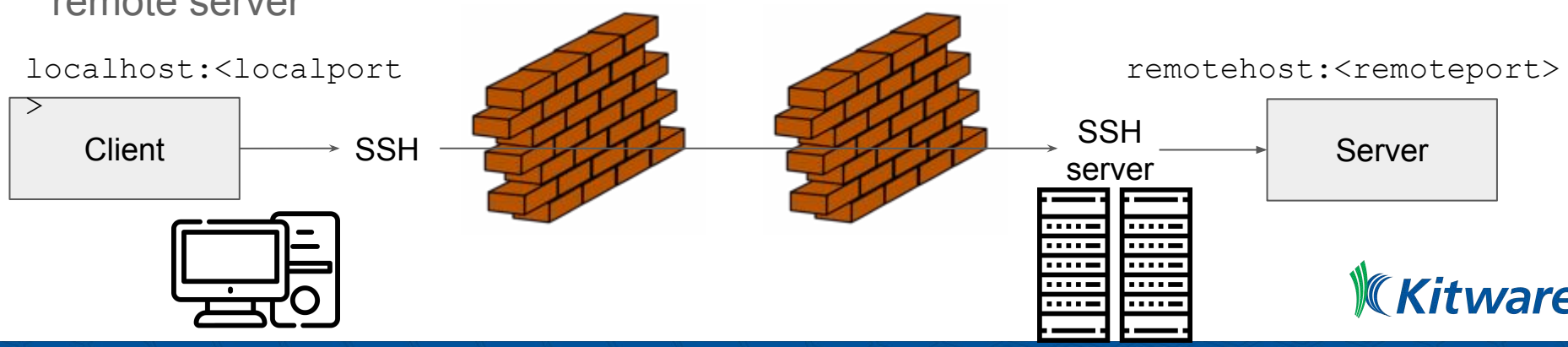
We can use this to create secure ParaView client/server connections through firewalls

SSH forward tunneling

```
> ssh -L <localport>:<remotehost>:<remoteport> <server>
```

SSH will listen on <localport> for a connection and forward the connection to <remotehost> on port <remoteport>

ParaView client connects to localhost:<localport> and SSH forwards this to the remote server



Kitware

SSH forward tunneling

Server side

```
localhost> ssh -L <localport>:remote:<remoteport> user@remote

remote> pvserver --server-port=<remoteport>
```

Client side

Add a new “Client / Server” (forward) connection where server **Host** is “localhost” and **Port** is <localport>

Kitware

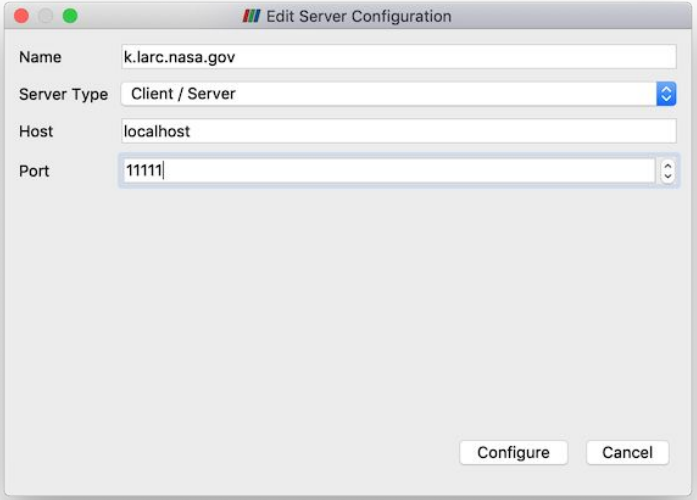
SSH forward tunneling - concrete example

Server side

```
localhost> ssh -L
11111:k.larc.nasa.gov:22222
user@k.larc.nasa.gov

user@k > pvserver \
  --server-port=22222
```

Client side



Kitware

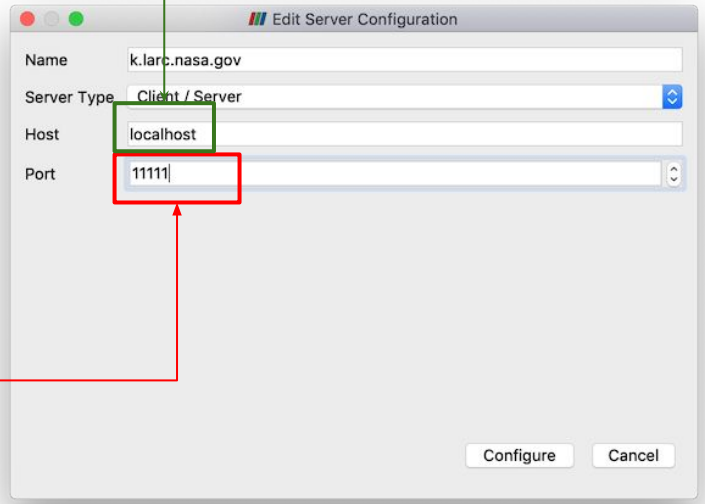
SSH forward tunneling - concrete example

Server side

Client side

```
localhost> ssh -L
11111:k.larc.nasa.gov:22222
user@k.larc.nasa.gov

user@k > pvserver \
  --server-port=22222
```

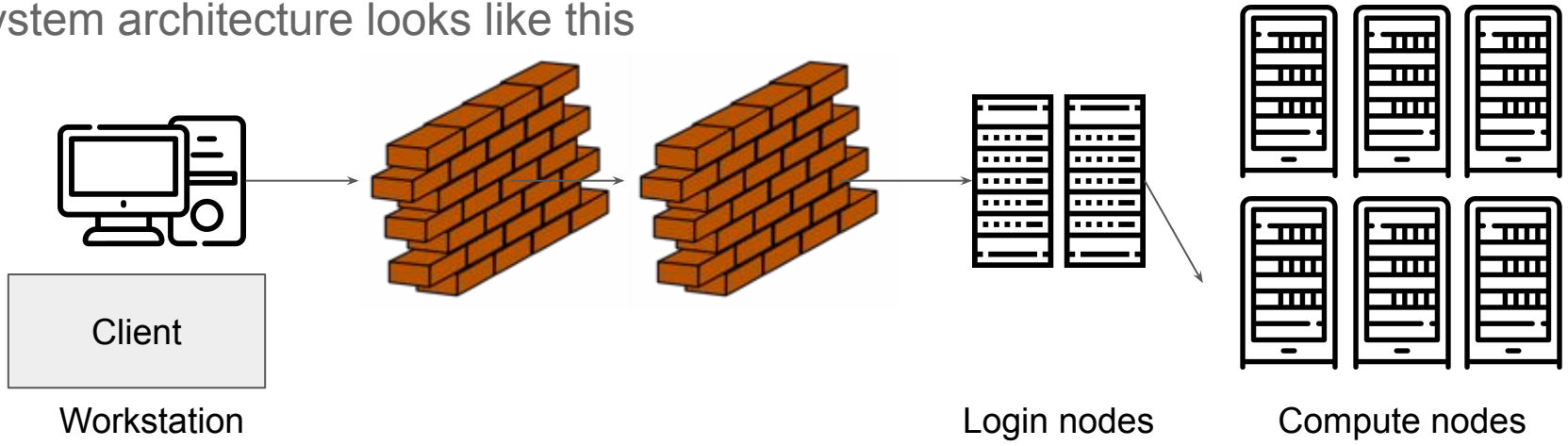


Kitware

Those pesky login nodes

Typically, there are dedicated login nodes separate from compute nodes

System architecture looks like this



How do we tunnel to a compute node?

Kitware

SSH forward tunneling with login nodes

Login nodes are not used for compute

- You need to submit a job to a queue
- You won’t know the DNS of the compute node

Forward tunneling relies on you knowing the hostname of the compute node

- How do we get that?

Kitware

SSH forward tunneling to compute node

Solution: use two terminals

Terminal 1: SSH into login node, queue job, get hostname of compute node

```
t1> ssh user@k.larc.nasa.gov
t1> qsub -I -l nodes=1 -q K4-standard
t1> hostname
compute-node21
```

Terminal 2: SSH into login node, set up forwarding from client to compute node

```
t2> ssh -L ZZZZZ:compute-node21:YYYYY user@k.larc.nasa.gov
```

Kitware

SSH forward tunneling to compute node

Launch `pvserver` on Terminal 1

```
t1> mpiexec -np <N> pvserver --server-port=YYYYY
```

With the ParaView client, connect to localhost on port `ZZZZZ`

This should connect the ParaView client through the SSH tunnel set up in Terminal 2 to the server on the compute node

Kitware

Two terminals is too many

Forward tunneling with two shells is cumbersome!

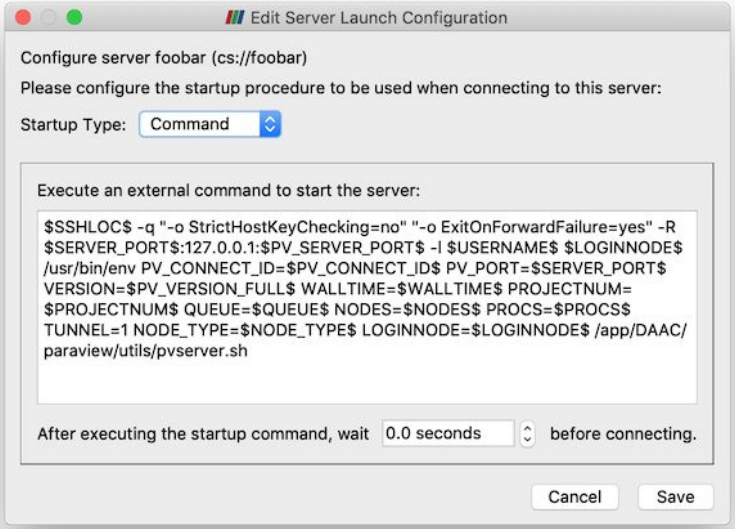
We can take care of launch of `pvserver` and connecting with a reverse SSH tunnel

Key is to use a Command **Startup Type** that starts `pvserver` when connection is made

Note variables such as $USERNAME$

These are defined in a .pvsc definition file that you can import to define server connections

Example:



Kitware

Example XML for previous connection

```
<Server name="Topaz (ERDC)" resource="csrc://localhost:11111">
    <CommandStartup>
      <Options>
        <Option name="USERNAME" label="Username" save="true">
          <String default=""/>
        </Option>
      </Options>
        …
      <Command exec="$SSHLOC$" timeout="0" delay="0">
        <Arguments>
          <Argument value="-q"/>
          <Argument value="-R"/>
          <Argument value="$SERVER_PORT$:127.0.0.1:$PV_SERVER_PORT$"/>
          <Argument value="-l"/>
          <Argument value="$USERNAME$"/>
        …
```

ParaView can download .pvsc file from URL and add connection configurations automatically

Configured servers are stored in .pvsc file in home directory:

%APPDATA%/ParaView/servers.pvsc
~/.config/ParaView/servers.pvsc

Kitware

Reverse connection

Connect button launches a program to SSH into and authenticate with login node and launches a script there. Client then waits for reverse connection from `pvserver` running on compute node. No separate SSH connection required.

The script:

Finds out login node's IP address

Submits a `pvserver` job through queue

`pvserver` job starts with `--reverse-connection` back to the waiting GUI client over the tunnel

Kitware

Reverse connection caveats

When login and compute nodes are separate, SSH daemon needs to have *AllowTCPForwarding* and *GatewayPorts* enabled

If not enabled, reverse connection cannot be made back to the client

Workaround is to use port forwarding utility such as `socat` on the login node listen for a connection on the compute node and forward it back to the client

Kitware

Reverse connection wrap-up

Offers a path to one-click connection to a remote cluster

May require assistance of a system administrator to set up, but then it is easy to make it available to all users

- Pass around a .pvsc file with configuration
- Download from a URL, for example: File->Connect->Fetch Servers

Kitware

Part 2
Remote visualization

Kitware

Loading data

Once a client and server are connected, we are ready to load data and visualize it

To open a data file, select the **File** -> **Open…** menu item

The file dialog shows files on the file system available to `pvserver`

You cannot load data on the local file system as long as the connection is alive

- Disconnect first to connect to the builtin server and load local data

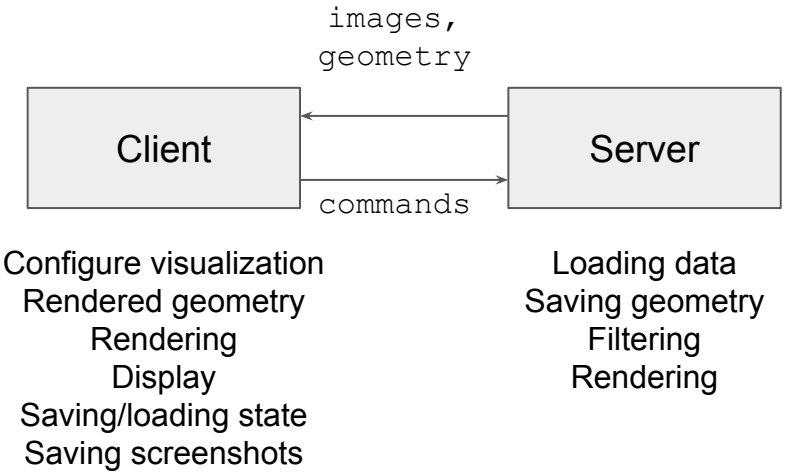
What happens where?

Data is loaded onto the `pvserver` processes

Filtering is performed on `pvserver`

Subsets of data can be sent to the client for local rendering

Rendering *can* be performed on `pvserver`, but it can also be done only on the client



Kitware

Where should rendering happen?

Client

Pros:

- Low latency from render request to render
- High frame rate

Cons:

- Fewer resources on client machine
- Network transfer time for geometry

Server

Pros:

- No geometry to transfer to client
- Can handle larger datasets

Cons:

- Higher latency, lower frame rate
- One image sent over network per render

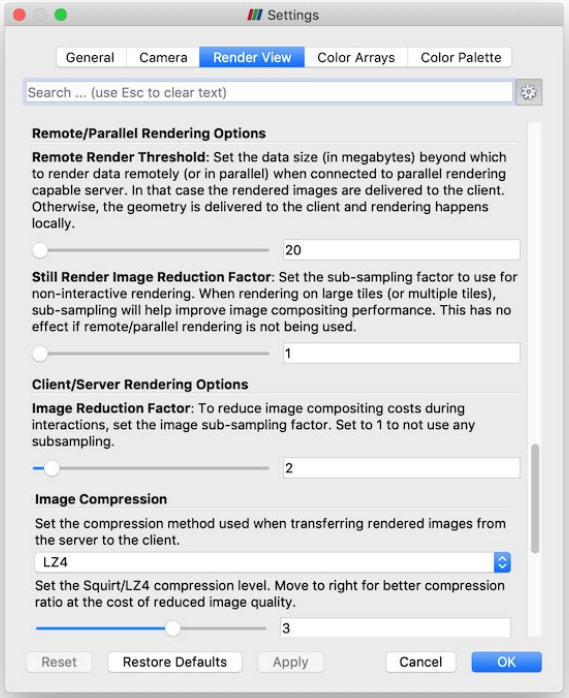
Kitware

Controlling where rendering happens

Usually, geometry is extracted from a dataset to be rendered

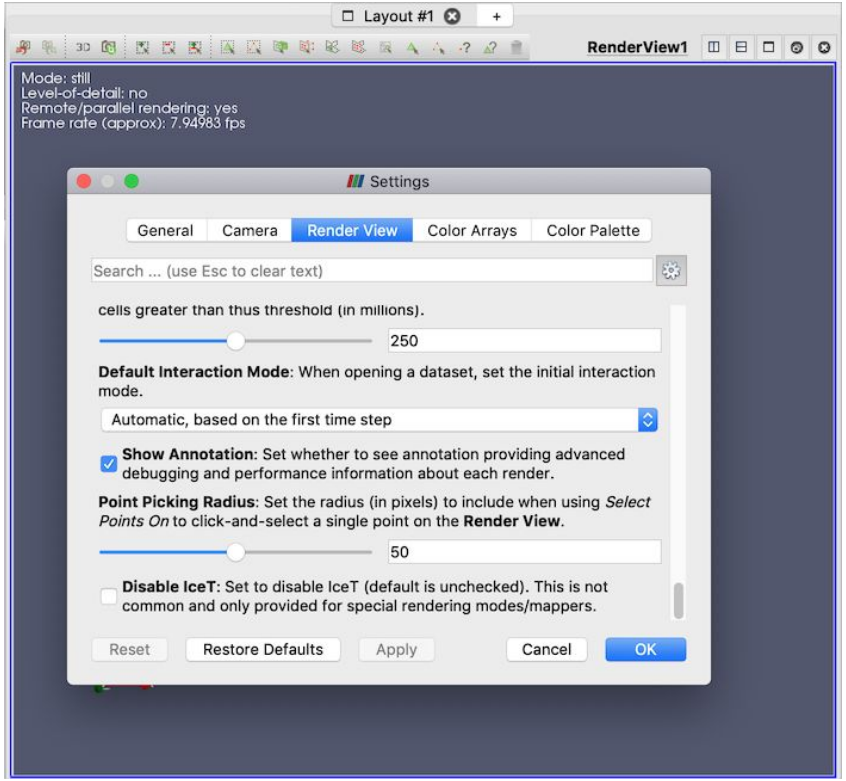
Remote Render Threshold determines where rendering happens

Available under **Edit** -> **Settings/ParaView** -> **Preferences**



Kitware

Verifying where rendering occurs



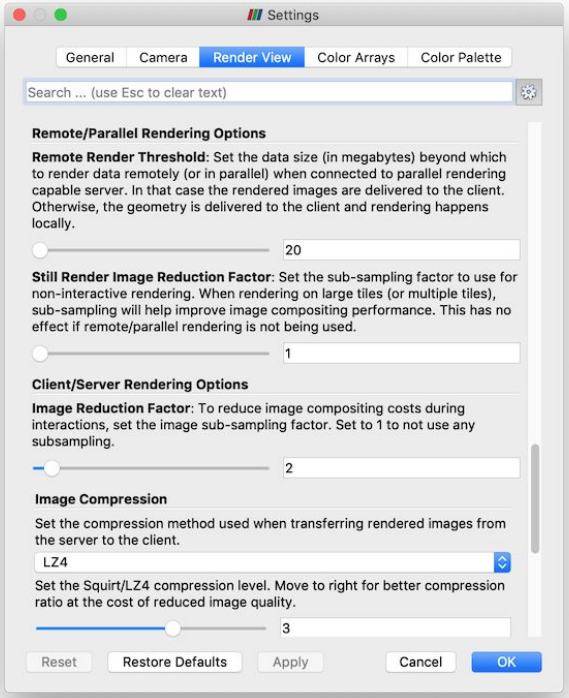
Kitware

Speeding up remote rendering - low bandwidth

Still Render Image Reduction Factor - subsample images when full quality rendering is requested

Image Reduction Factor - subsample images generated during interactive rendering

Image Compression - compression algorithm and amount of compression to use



Kitware

Speeding up remote rendering - high latency

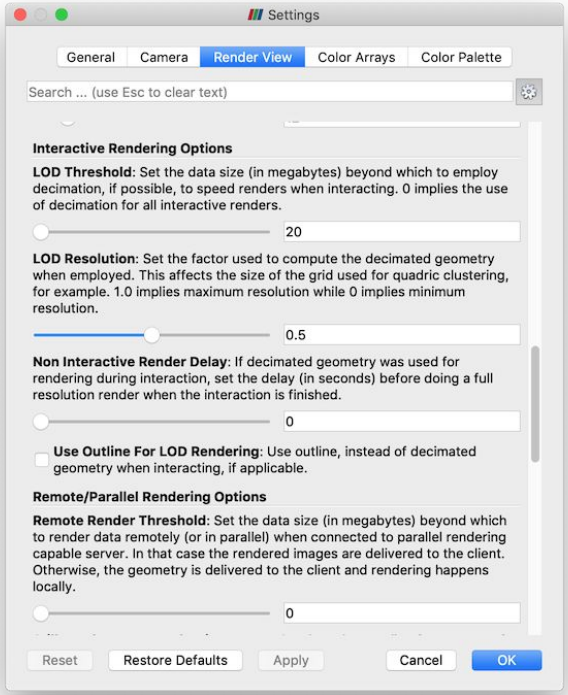
Play with the **LOD Threshold** and **LOD Resolution** to reduce geometry sent to client.

Turn on **Use Outline for LOD Rendering** if decimated geometry too big for client.

Try increasing **Non Interactive Render Delay**

Turn up **Remote Render Threshold**

Allow more data to go to client



Kitware

Remote visualization with Python

`pvpython` is a ParaView client application that can generate images from a Python script

To connect to a pvserver on a remote system, use the Connect() function

```
>>> Connect("server.host", 11111)
```

Once connection is established, Python commands can be run to create filters and generate images on the server

Kitware

Questions?

Kitware