An introduction to XENON

Jason Maassen, Arnold Kuzniar, Jurriaan Spaaks, Stefan Verhoeven, Johan Hidding, ....
We are an institute that provides funding + RSEs to research projects.
so far: ~130 projects
(on many different topics)

**Humanities & Social Sciences**
- incl. SMART cities, text analysis, creative technologies

**Physics & Beyond**
- incl. astronomy, high-energy physics, advanced materials

**Sustainability & Environment**
- incl. climate, ecology, energy, logistics, water management

**Life Sciences & eHealth**
- incl. bio-imaging, next generation sequencing, molecules
One of our goals: software re-use

We try to re-use research software between projects
  spreads cost of maintenance and development
  re-use expertise of the engineers
  increases user community
  increases sustainability → more bang for buck!

Works best for software low on the software stack
  generic tools are easier to reuse
Examples applications

A recurring theme in many of our projects is easy access to (remote) compute and storage.
What is the problem?

For many applications we need to copy data and submit jobs to remote systems. There are many ways to do this:

- FTP, SFTP, WebDAV, S3, Hadoop, GridFTP, iRods, ...
- Slurm, GridEngine, PBS, Torque, Amazon-Batch, ...

There are libraries and CLIs for each of these, however:
- you have to figure out how they work
- picking one will lock-in your solution
- using all of them is way too much work!
xenon
(and friends)

A collection of libraries and tools that provide easy access to (remote) compute and storage resources

Used via-via in many other tools and projects.
xenon
(and friends)

Core packages at:
https://github.com/xenon-middleware
How does xenon help?

Xenon offers simple programming interfaces to access remote file systems and schedulers.

Adaptors implement the functionality for different backends.

A CLI allows for easy use in scripts.
xenon (the library)

Target audience is developers creating tools and (other) libraries.

Programming interface is kept simple, just focus on basic tasks:

- submit a job
- copy files

Good enough for 90% of the cases
xenon-cli
(the command line tool)

Target audience is users creating scripts and workflows

Offers uniform syntax to use different platforms with focus on basic tasks:

- submit a computation
- copy files

Good enough for 90% of the cases
Combining filesystems and schedulers

So far, we've used xenon to manipulate files on the local filesystem, and to run system executables on the remote machine. In typical usage, however, you would use xenon to run executables or scripts of your own, which means that we need to upload such files from the local system to the remote system.

A typical workflow may thus look like this:

1. upload input file(s)
2. submit job
3. download generated output file(s)

Use an editor to create a file `sleep.sh` with the following contents (the virtual machine comes with a bunch of editors like `gedit`, `leafpad`, and `nano`, but you can install a different editor from the repositories if you like):

```
#!/usr/bin/env bash

echo "date": went to bed

echo "date": woke up"
```

You can test if your file is correct by:

```
# last argument is the sleep duration in seconds
bash sleep.sh 5
```

We need to upload `sleep.sh` to the remote machine. We can't use `xenon filesystem` file like we did before, because we're copying between file systems, so let's look at what other options are available:

```
xenon filesystem --help

# let's try sftp protocol
xenon filesystem sftp --help

# we're interested in 'upload' for now
xenon filesystem sftp upload --help
```

We'll also need to tell xenon what location we want to connect to, and what credentials to use. The SLURM Docker container we used before is accessible via SFTP using the same location, username and password as before, so let's use that:

```
Bash

[Java]

xenon filesystem sftp --location Host=192.168.0.1:2222 --username xenon --password
upload /home/travis/sleep.sh /home/xenon/sleep.sh

[Python]

# step 1: upload input file(s)
xenon filesystem sftp --location Host=192.168.0.1:2222 --username xenon --password
upload /home/travis/sleep.sh /home/xenon/sleep.sh

# step 2: submit job
```

Now that the script is in place, we can submit a bash job using xenon scheduler `slurm submit` like before, taking the newly uploaded file as input to `bash`, and using a sleep duration of 60 seconds:

```
# step 2: submit job
```

https://xenon-tutorial.readthedocs.io

Explains how to use the
- Python API
- Java API
- Command Line Interface

Provide Linux VM image with
- pre-installed xenon
- docker containers (slurm, sftp)
<table>
<thead>
<tr>
<th>Bash</th>
<th>Java</th>
<th>Python</th>
</tr>
</thead>
</table>

```
xenon filesystem file list /home/travis/fixtures
```
```java
package nl.esiencecenter.xenon.tutorial;

import nl.esiencecenter.xenon.filesystems.FileSystem;
import nl.esiencecenter.xenon.filesystems.Path;
import nl.esiencecenter.xenon.filesystems.PathAttributes;

public class DirectoryListing {
    public static void main(String[] args) throws Exception {
        String adaptor = "file";
        FileSystem filesystem = FileSystem.create(adaptor);
        Path directory = new Path("/home/travis/fixtures");
        Boolean recursive = false;
        Iterable<PathAttributes> listing = filesystem.list(directory, recursive);
        for (PathAttributes elem : listing) {
            if (!elem.isHidden()) {
                System.out.println(elem.getPath());
            }
        }
    }
}
```
```python
import xenon
from xenon import Path, FileSystem

def run_example():
    xenon.init()

    filesystem = FileSystem.create(adaptor='file')
    path = Path("/home/travis/fixtures")
    listing = filesystem.list(path, recursive=False)

    for entry in listing:
        if not entry.path.is_hidden():
            print(entry.path)

    filesystem.close()

if __name__ == '__main__':
    run_example()
```
Portable HPC workflows based on Snakemake + Conda + Xenon

Arnold Kuzniar

netherlands eScience center
Googling the cancer genome project

Identification and prioritization of cancer-causing structural variants (SVs)

UMCU team
Jeroen de Ridder (PI)
Wigard Kloosterman
Luca Santuari
Carl Shneider

eScience team
Lars Ridder (coordinator)
Arnold Kuzniar
Sonja Georgievska
Jason Maassen
Stefan Verhoeven
Challenge: reliable detection of SVs
sv-callers workflow

Combines existing SV detection tools into a portable HPC workflow

<table>
<thead>
<tr>
<th>Software</th>
<th>Implementation</th>
<th>Parallelism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manta</td>
<td>C++, Python</td>
<td>pyFlow(^1) tasks, SIMD(^2)</td>
</tr>
<tr>
<td>DELLY</td>
<td>C++</td>
<td>OpenMP(^3) threads</td>
</tr>
<tr>
<td>LUMPY</td>
<td>C/C++, Python</td>
<td>not supported</td>
</tr>
<tr>
<td>GRIDSS</td>
<td>Java, R, Python</td>
<td>Java threads, SIMD(^2)</td>
</tr>
</tbody>
</table>
sv-callers workflow

Conda is used to enable automatic install of the tools on the target system
sv-callers workflow

Snakemake is used to define the overall workflow
sv-callers workflow

Snakemake then uses xenon to deploy on the available resources
Snakemake → Xenon → GridEngine

```bash
snakemake -C echo_run=1 mode=p
enable_callers="['manta','delly','lumpy','gridss']"
--use-conda --latency-wait 30 --jobs 9
--cluster 'xenon scheduler gridengine
--location local:// submit --name smk.{rule} --inherit-env
--procs-per-node {threads} --start-single-process
--max-run-time 1 --max-memory {resources.mem_mb}
--working-directory . --stderr stderr-%j.log
--stdout stdout-%j.log' &>smk.log&
```
Snakemake → Xenon → Slurm

```
snakemake -C echo_run=1 mode=p
enable_callers="['manta','delly','lumpy','gridss']"
--use-conda --latency-wait 30 --jobs 9
--cluster 'xenon scheduler slurm'
--location local:// submit --name smk.{rule} --inherit-env
--procs-per-node {threads} --start-single-process
--max-run-time 1 --max-memory {resources.mem_mb}
--working-directory . --stderr stderr-%j.log
--stdout stdout-%j.log' &>smk.log&
```
Results

(A) Germline and (B) somatic SVs detected in the benchmark and in the cell lines samples, respectively, using Manta, DELLY, LUMPY and GRIDSS. Most SVs are caller-specific, followed by SVs common to three of the four callers. SVs detected by the callers were filtered and merged into one set (see the Methods section). Note: The Venn diagrams include the largest GRIDSS sets as the GRIDSS output varies slightly each run using the same input. Fig. S1-2 show the comparisons across sample copies.

“sv-callers: a highly portable parallel workflow for structural variant detection in whole-genome sequence data”

Arnold Kuzniar, Jason Maassen, Stefan Verhoeven, Luca Santuari, Carl Shneider, Wigard P. Kloosterman, Jeroen de Ridder

Accepted for publication in PeerJ

https://github.com/GooglingTheCancerGenome/sv-callers
Xenon Roadmap

almost done: GridFTP, amazon-batch

in progress: azure-batch

ideas: iRODS (?), google storage (?), ...
      language bindings for C++, C#, Go, etc.
      better integration in snakemake,

suggestions are welcome!
Thanks!

https://github.com/xenon-middleware
https://xenon-tutorial.readthedocs.io
https://github.com/GooglingTheCancerGenome/sv-callers

xenon@esciencecenter.nl
j.maassen@esciencecenter.nl

www.esciencecenter.nl
Interested in Research Software?

The Netherlands eScience Center is the Dutch national center of excellence for the development and application of research software to advance academic research.

Join the team!

✉️ n.renaud@esciencecenter.nl

📞 +31 (0)20 460 4770
🌐 www.esciencecenter.nl
📖 blog.esciencecenter.nl
Extra slides
SIM-CITY

Goal: decision support for urban social economic complexity

Run urban planning simulations from an interactive user interface.

Example: location of fire stations

Simulations must be started on a remote compute cluster.
eMetabolomics

Goal: computer assisted metabolite profiling.

Determine how compounds are metabolized in the human body based on mass spectrometer data and metabolism rules.

Requires starting analysis computations on a server containing the metabolism database.
AMUSE / OMUSE

Goal: combine existing physics simulations into more complex models a simple python framework.

Originally for astrophysics (AMUSE), but now also for ocean modelling (OMUSE).

Requires starting simulations on multiple compute clusters and controlling them remotely.