UCSD-NT x SDSC Expanse

Ricky Mok and Max Gao (CAIDA/UC San Diego)

February 13, 2025

UCSD-NT Datasets

Currently UCSD-NT offers four modes of data format

1. Timeseries (https://explore.stardust.caida.org)
2. FlowTuple (https://www.caida.org/projects/stardust/docs/data/flowtuple/)
   a. Version 4 since 2019
   b. Version 3 prior to 2019
3. Pcap
   a. On-premise for the last 30 days
   b. Archived pcaps (upon request)
4. Live capture in a VM

Doc: https://www.caida.org/projects/network_telescope/docs/

UCSD-NT Datasets

Currently UCSD-NT offers four modes of data format

1. Timeseries (https://explore.stardust.caida.org)
2. FlowTuple (https://www.caida.org/projects/stardust/docs/data/flowtuple/)
   a. Version 4 since 2019
   b. Version 3 prior to 2019
3. Pcap
   a. On-premise for the last 30 days
   b. Archived pcaps (upon request)
4. Live capture in a VM

SDSC Expanse

A high-performance cluster consists of 728 compute nodes (>90k CPU cores) and 52 GPU nodes (~200 GPUs).

- Attach Lustre storage to (temporary) store data for processing and/or output
- Use the slurm workload manager to schedule and manage compute jobs
- Request the access and credits via ACCESS-CI (assume you all have an account)

Learning objectives of this tutorial

1. Basic use of SDSC Expanse
2. Access UCSD-NT data from Expanse
3. Perform interactive and batch data analysis with 4 examples
   a. Interactive FlowTuple analysis with Jupyter Notebook
   b. Batch FlowTuple analysis with pyspark
   c. Batch Pcap data analysis using Golang with concurrency
   d. Parallelize Pcap data analysis using GNU parallel

What do you need?

Your have an ACCESS-CI account

An ACCESS-CI allocation (We sorted this out if you sent me your ACCESS-CI username before the tutorial)

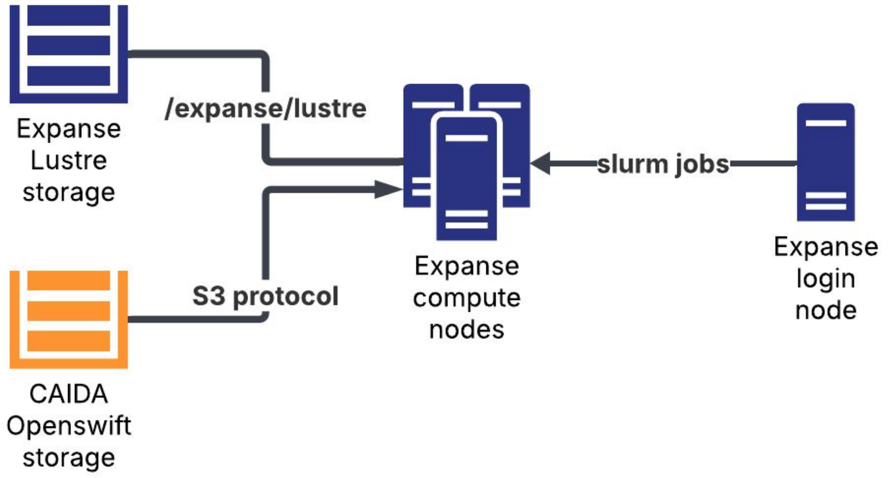
- If you don’t have them, it is probably too late. But we can still set that up today, so you can try the code when you return home.

Clone the tutorial Github repo to your Expanse home directory: https://github.com/CAIDA/telescope_expanse_tutorial

Architecture overview

Users interact with the login node to deploy jobs, and run actual tasks on compute nodes

!!! Do not run any compute/data intensive task on the login node !!!



Login to SDSC Expanse

Login node hostname: login.expanse.sdsc.edu

Username: <Access user name>, Password: <Access password>

Upon login, you can upload your private ssh key for key login and clone the git repository

```
[laptop]$ ssh expanse
[login01]$ git clone https://github.com/CAIDA/telescope_expanse_tutorial
[login01]$ expanse-client user
```

Storage

Lustre storage (won’t go away as the job completed)

- Project storage (persistent, but no guarantee): `/expanse/lustre/projects/csd939/<username>`
- User scratch space (files will be purged in 90 days) : `/expanse/lustre/scratch/<username>`

On-node scratch space (local SSD, but will be erased as the job completed)

```
/scratch/<username>
```

Partitions

Expanse uses “partitions” to accommodate different resource requirements

These three partitions are commonly used, and have different limits in maximum time, nodes, jobs, charges, and wait time.

`debug` - Max 30 mins runtime. For testing small jobs

`shared` - Max 48 hrs runtime for jobs using a single node and <128 cores

`compute` - Max 48 hrs runtime. Exclusive access to one or more compute nodes

Refer to “Running Jobs on Expanse” section under https://www.sdsc.edu/systems/expanse/user_guide.html

Slurm basic

Slurm scripts are shell scripts embedded with Slurm specific configuration.

```
#SBATCH --job-name=spark-pi #your job name
#SBATCH --partition=debug #Expanse partition
#SBATCH --account=csd939 #project id to charge
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1              Resource settings
#SBATCH --cpus-per-task=8
#SBATCH --mem-per-cpu=1000
#SBATCH --time=00:30:00 #maximum task runtime
#SBATCH --output="joblogs/slurm-%A_%a.out" #stdout/stderr
#SBATCH --mail-type=BEGIN
#SBATCH --mail-user=my@email.com           Email notification
```

Github repo organization

You can find scripts used in this tutorial here

https://github.com/CAIDA/telescope_expanse_tutorial/

`slurm_scripts` - Slurm scripts for deploying all examples

We provide a Golang library specific to UCST-NT with two examples

https://github.com/CAIDA/goucsdnt

Example 1

Goal: Launch jupyter notebook to perform *interactive* analysis on Expanse

- Count packets targeted to different destination /16s in the telescope

Steps overview

1. Schedule the slurm job
2. Check the status and wait for your allocation
3. Start jupyter notebook on the assigned compute node
4. Run the analysis in the notebook

Scheduling your first job

Change account name to csd939 in slurm_scripts/compute-1x8_8G-debug.slurm

Line 4: `#SBATCH --account=csd939`

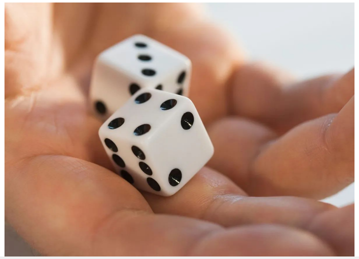
Schedule the job

```
[login01]$ sbatch slurm_scripts/compute-1x8_8G-debug.slurm
```

Expected output

```
Submitted batch job 36745199 ← your job id
```

Check and wait



Now you need to roll your dice…

Use `squeue -u <username>` to check the status of your job

```
[login02 telescope_expanse_tutorial]$ squeue -u <username>
         JOBID   PARTITION     NAME     USER          ST     TIME  NODES NODELIST(REASON)
      36747293    shared       spark-pi  <username> PD      0:00     1        (Priority)
```

PD status means there is no available resource at this time due to Priority. You just need to wait.

If you request a node in “compute” partition, it could take hours.

Tips: Set the mail-address and mail-type in your slurm script. You will receive an email notification when the resource is available.

Your task is running!

R status means your task is running on the node listed under NODELIST.

```
   JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
36747289     debug spark-pi     kmok  R       0:01      1 exp-9-55
```

You can now login to the node from the Internet and start a ssh tunnel to connect to your jupyter notebook

```
[laptop]$ssh -i {SSH_KEY} -L 9000:localhost:9000
{USER}@exp-9-55.expanse.sdsc.edu
```

Start the example notebook

We prepared a start-up script for you in the github repo

```
[user@exp-9-55 ~]$ cd telescope_expanse_tutorial/example_notebooks/
[user@exp-9-55 example_notebooks]$ bash run_notebook.sh
```

…<typical jupyter notebook output>...

    Or copy and paste one of these URLs:

        http://localhost:9000/lab?token=<token>

…

Now you can start the browser and access your notebook

Jupyter notebook example

Set up Spark environment

Load FlowTuple files directly from CAIDA’s swift storage

Apply filters to obtain targeted IPs, protocols, etc

Count the unique IPs and packets to different telescope subnets

Behind the curtain

launch-spark.sh

- Copy pre-compiled spark and hadoop binary to your scratch directory and generate configuration files

bootstrap_env.sh

- Setup python virtual environment and install required python modules

When to use this approach?

Interactive interface to debug the code

Mid-scale ad-hoc FlowTuple analysis

As the notebook is running on local scratch, remember to save your notebook to persistent storage before the runtime expires or when you completed your work!

Example 2

Batch analysis for FlowTuple data

- Strip down version of example 1
- Perform the same analysis with a python script (example_ft_batch/count-dstnet.py)

Run

```
[login01]$sbatch slurm_scripts/compute-1x8_8G-batch-debug.slurm
```

Similarly, you can check the job status using

```
[login01]$squeue -u <username>
```

So now?

Go get a cup of coffee…

Your task is scheduled to run.

Once it is executed, you will find the results in `/expanse/lustre/projects/{PROJECT}/{USER}/` (line 43 example_ft_batch/count-dstnet.py).

Exercise

Can you modify the slurm script, so you receive an email notification upon the completion of the task?

```
#SBATCH --mail-user=<your email>
#SBATCH --mail-type=END
```

Use cases

You have a “bug-free” code that you tested in the interactive mode, and apply it to long time period of data.

Pros:

- Your analysis can use all the resources (not for running notebook)

Cons:

- Debugging could be harder and more time consuming

Example 3 and 4

Python may not be your programming language (slow, hard to parallelize, …)

You may already have your code written in other languages or a docker container built on your own machine

Further, Expanse does not have the binary latest software stack

- E.g., precompiled golang version on Expanse is 1.15 (latest is 1.23)

Goal: Execute a **docker container** on Expanse

Build your container

1. Setup a Docker container
2. Upload your container to Dockerhub
3. Convert the docker container into singularity container
4. Upload the singularity container to Expanse lustre storage
5. Schedule the execution of the container on Expanse compute node

Detail steps can be found in the readme in example_golang_docker folder.
https://github.com/CAIDA/telescope_expanse_tutorial/tree/main/example_golang_docker

Just like any other cooking shows

Here’s the one I prepared earlier :)

```
/expanse/lustre/projects/csd939/kmok/
shared_data/expanse-uscdnt-golang.sif
```

What does the sample code do?

The sample code performs (https://github.com/CAIDA/goucsdnt/blob/master/cmd/analyzepcaps3/analyzepcaps3.go)

1. Stream pcap files directly from CAIDA's Swift object storage according to start (`-s`) /end time (`-e`) or a file name (`-f`). Stop the processing after reading (`-c`) number of packets.
2. Parallelize the processing into (`-w`) number of workers if the files are selected using time.
3. Count the number of packets with Mirai malware signature (Dest. IP = TCP Seq number) from each source IP.
4. Output the counts into CSV files

Example 3

The example selects pcap files by a time range. The program (`analyzepcaps3.go`) process pcap files concurrently using goroutines.

- I.e., you handled the concurrency problem within your code

Executing the program is simple in the slurm script

```
singularity exec --env-file /expanse/lustre/projects/csd939/kmok/.ucsdnts3.env \

--bind /expanse/lustre/projects/csd939/${USER}/golang_container_output:/output
,/tmp,/scratch \

    /expanse/lustre/projects/csd939/kmok/shared_data/expanse-uscdnt-golang.sif \

    /go/bin/ucsdntexample -o /output -c 100 -e "2025-01-25T02:59:59Z"
```

Env var

Vol Mount

Img location

command

Let’s try it

In the login node,

```
$ sbatch slurm_scripts/container-1x8_8G-batch-debug.slurm
```

After the completion of the task , you can find the output (csv) files in

```
/expanse/lustre/projects/csd939/${USER}/golang_container_output
```

Do you remember how to check your job status?

Example 4

We will run the same program, but this time we do not use the goroutine approach for concurrency (emulating single threaded programs).

Instead, we use GNU parallel to achieve some levels of concurrency.

The final example

Copy the file list from slurm_scripts/filelists.csv to /expanse/lustre/projects/csd939/kmok/golang_container_output/

Execute the slurm script

```
[login01]$sbatch
slurm_scripts/container-1x8_8G-parallel-batch-debug.slurm
```

Difference between Example 3 and 4

The last line the slurm script in example 4.

```
    bash -c "cat \"/output/filelists.csv\" |
/usr/bin/parallel -j 5 \"/go/bin/ucsdntexample -o /output -c
100 -f {}\""
```

Each line in the filelists.csv file will be used as input argument of the program.

`-j 5` limits the maximum number of parallel jobs to 5

Environment for example 4 may be slightly harder to set up, but still manageable

How about processing the entire pcap?

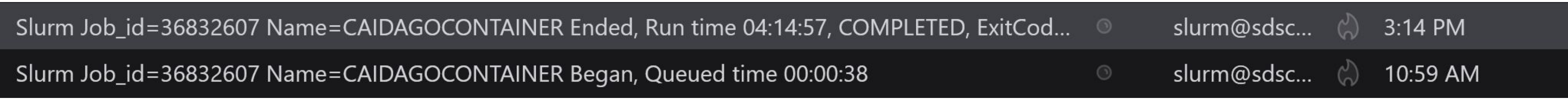
Example 3 and 4 only check the first 100 packets (`-c 100`). Removing this input argument will process the entire pcap.

Modified Example 3:

```
/go/bin/ucsdntexample -o /output -w 5 -e "2025-01-25T06:59:59Z"
```

This will process 7 hours of pcaps with 5 goroutines

The runtime was about 4 hr 15 mins on one compute node.



```
Slurm Job_id=36832607 Name=CAIDAGOCONTAINER Ended, Run time 04:14:57, COMPLETED, ExitCod...   slurm@sdsc...   3:14 PM
Slurm Job_id=36832607 Name=CAIDAGOCONTAINER Began, Queued time 00:00:38                       slurm@sdsc...   10:59 AM
```

Runtime check

Start time: 10:59am

```
-rw-r--r-- 1 kmok csd939 331755 Feb 10 13:26 ucsd-nt.1737763200.csv
-rw-r--r-- 1 kmok csd939 334242 Feb 10 13:21 ucsd-nt.1737766800.csv
-rw-r--r-- 1 kmok csd939 334670 Feb 10 13:26 ucsd-nt.1737770400.csv
-rw-r--r-- 1 kmok csd939 336407 Feb 10 13:25 ucsd-nt.1737774000.csv
-rw-r--r-- 1 kmok csd939 334796 Feb 10 13:23 ucsd-nt.1737777600.csv
-rw-r--r-- 1 kmok csd939 332974 Feb 10 15:14 ucsd-nt.1737781200.csv
-rw-r--r-- 1 kmok csd939 329890 Feb 10 15:08 ucsd-nt.1737784800.csv
```

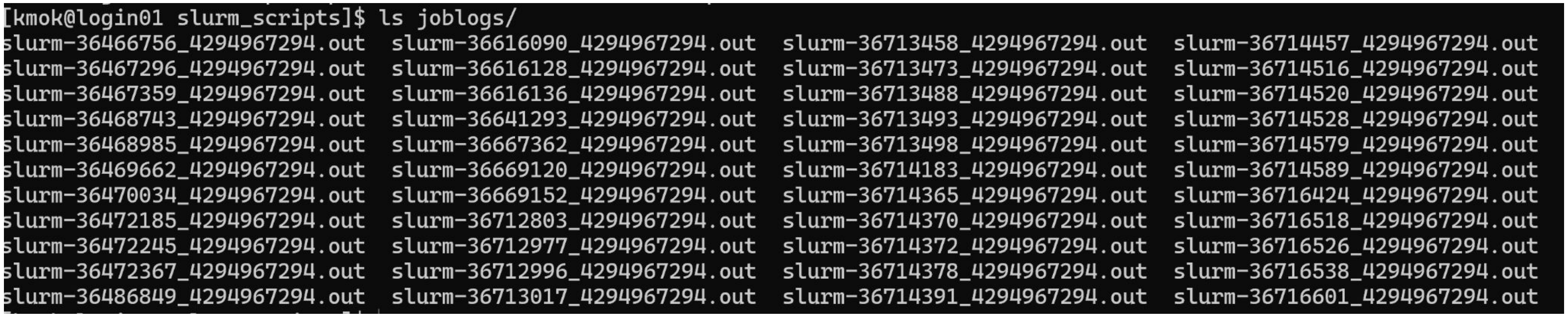
First batch (5 files): ~2hr 25 mins
Second batch (2 files): ~1 hr 55 mins

Debugging your job

In the slurm script:

```
#SBATCH --output="joblogs/slurm-%A_%a.out"
```

The stdout/stderr will be saved in joblogs directory. `%A` is the job ID, `%a` is the job array index (not used in our examples).



```
[kmok@login01 slurm_scripts]$ ls joblogs/
slurm-36466756_4294967294.out  slurm-36616090_4294967294.out  slurm-36713458_4294967294.out  slurm-36714457_4294967294.out
slurm-36467296_4294967294.out  slurm-36616128_4294967294.out  slurm-36713473_4294967294.out  slurm-36714516_4294967294.out
slurm-36467359_4294967294.out  slurm-36616136_4294967294.out  slurm-36713488_4294967294.out  slurm-36714520_4294967294.out
slurm-36468743_4294967294.out  slurm-36641293_4294967294.out  slurm-36713493_4294967294.out  slurm-36714528_4294967294.out
slurm-36468985_4294967294.out  slurm-36667362_4294967294.out  slurm-36713498_4294967294.out  slurm-36714579_4294967294.out
slurm-36469662_4294967294.out  slurm-36669120_4294967294.out  slurm-36714183_4294967294.out  slurm-36714589_4294967294.out
slurm-36470034_4294967294.out  slurm-36669152_4294967294.out  slurm-36714365_4294967294.out  slurm-36716424_4294967294.out
slurm-36472185_4294967294.out  slurm-36712803_4294967294.out  slurm-36714370_4294967294.out  slurm-36716518_4294967294.out
slurm-36472245_4294967294.out  slurm-36712977_4294967294.out  slurm-36714372_4294967294.out  slurm-36716526_4294967294.out
slurm-36472367_4294967294.out  slurm-36712996_4294967294.out  slurm-36714378_4294967294.out  slurm-36716538_4294967294.out
slurm-36486849_4294967294.out  slurm-36713017_4294967294.out  slurm-36714391_4294967294.out  slurm-36716601_4294967294.out
```

Hold on

Good practices to follow:

The main bottleneck of such setup is usually the network link between CAIDA's storage and Expanse (just 10Gbps) and/or the processing capability of the storage cluster

- Processing many (>=10 pcap/>50 FlowTuple) files at the same time **will not** reduce your runtime.
- **We recommend to stream <=5 pcap/<25 FlowTuple files at the same time**

You can transfer (limited amount of) files to lustre storage if you plan to re-run them

Which dataset do you need?

I want to know …

- Who (source IPs/ASes/countries) sent traffic to UCSD-NT
- The services (by dest port) that scanners were interested in
- The protocols/ dest. port that a particular (set of) scanners targeted

→ FlowTuple

- Inspect packet-level fingerprints (complete headers/payload)
- Packet timing

→ Pcap

Service Unit $$

Ref: Job Charging section
https://www.sdsc.edu/systems/expanse/user_guide.html

1 SU = one compute core utilizing less than or equal to 2G of data for one hour, or 1 GPU using less than 92G of data for 1 hour.

For example, you requested 1 node on compute partition (128 CPU) for 1 hour. 128 SU will be charged.

Researchers at US-based institutions can apply for ACCESS-CI allocation (starting at 200k). Oversea ones may require US collaborators to obtain allocation

Enjoy

You can build your analysis now (or later). We will keep the allocation and the access key until the end of February.

Be nice. Don't drain all the SUs in the tutorial allocation :)

If you want to use the telescope data for your research

- Fill CAIDA data request form for your own S3 credential
- Apply your own ACCESS-CI allocation

Updates

```
#SBATCH -partition=shared
```

A newer way to span up jupyter notebook on Expanse

https://github.com/sdsc/galyleo

We will update the Github repository for UCSD-NT specific examples