WLCG Environmental Sustainability WS (CERN)

Credits: J. Flix / K. Fàbrega

Date: 11/12/2024

In [1]:
```
import numpy as np
import pandas as pd
import os
import matplotlib as mpl
import matplotlib.pyplot as plt
from time import strftime, localtime
import sys
import glob
import re
```

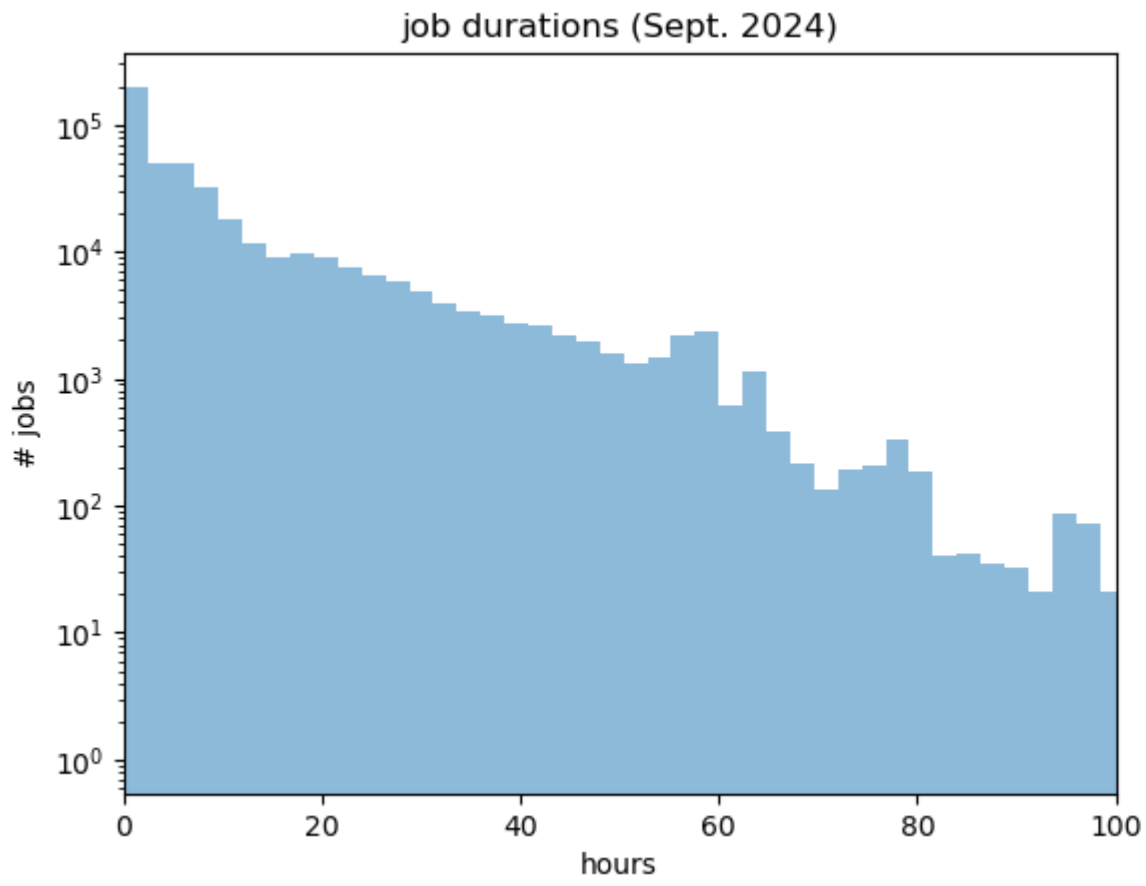
Read September 2024 data from 5 HTCondor-CE's + local submits at PIC

In [2]:
```
df = pd.read_pickle("./df_sept2024.pkl")
df.sort_values('timestamp', ascending=True, inplace=True)
```

Plot job durations

The PIC local HTCondor config allows for jobs to extend up to 100h

In [4]:
```
plt.hist(df['jobduration']/3600,bins=100,log=True, alpha=0.5)
plt.xlabel("hours")
plt.ylabel("# jobs")
plt.title("job durations (Sept. 2024)")
plt.xlim(0,100)
plt.show()
```



Resemble PIC farm utilization

From the job timestamps (start_time and end_time), and the number of cores used per job, we can resemble the PIC farm utilization.

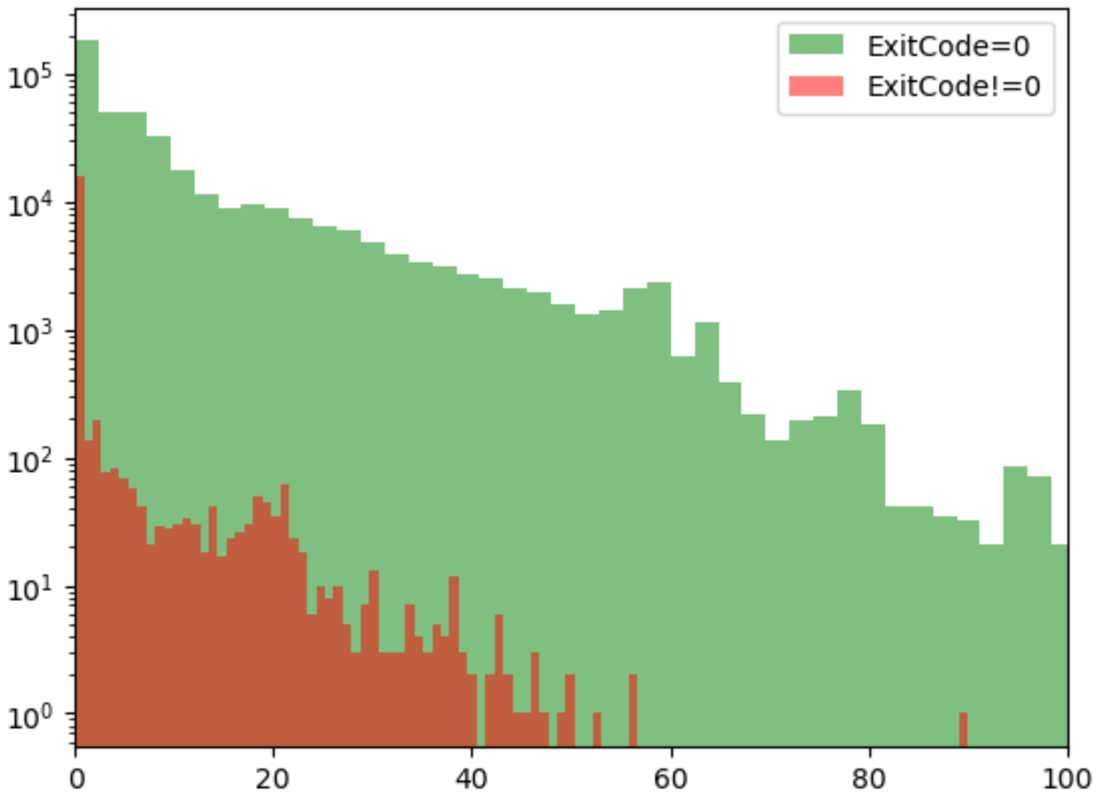
Querying in intervals of 60 seconds for the whole period, we can compute how many jobs were running at a particular time (i.e. 'startdate' <= t & 'enddate' > t). Of course, we miss those jobs that have a duration of < 60 seconds. There are ~6% of jobs failing there. Of course, the 60 seconds default can be lowered, but the computational time increase. We use 60 seconds for this live demo.

In [5]:
```
print("# jobs:", df.shape[0])
print("# jobs (<60 s):", df[df['jobduration']<60].shape[0])
print("Fraction (%):", 100.*df[df['jobduration']<60].shape[0]/df.shape[0]
```

```
# jobs: 445455
# jobs (<60 s): 26094
Fraction (%): 5.857830757315554
```

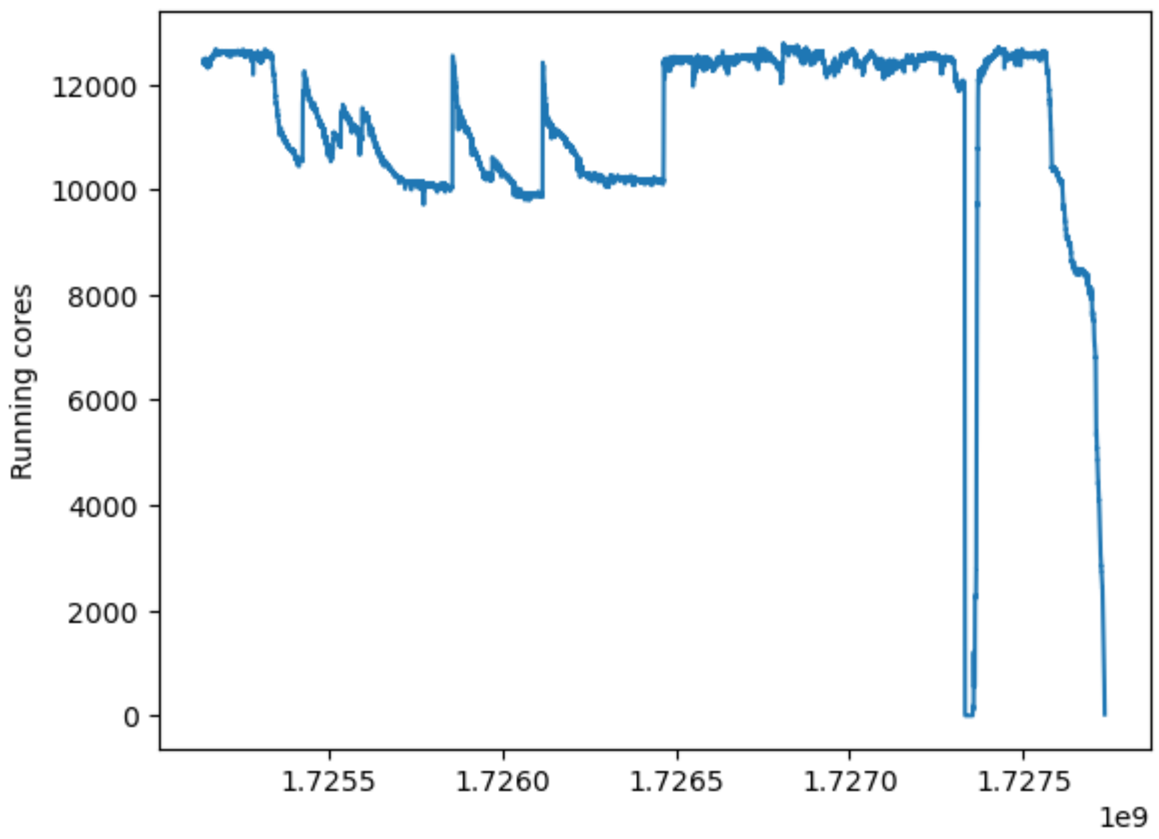
Indeed the jobs we are losing are those that mostly fail...

In [6]:
```
plt.hist(df['jobduration'][df['ExitCode'] == "0"]/3600,bins=100, color="g
plt.hist(df['jobduration'][df['ExitCode'] != "0"]/3600,bins=100, color="r
plt.xlim(0,100)
plt.legend()
plt.show()
```



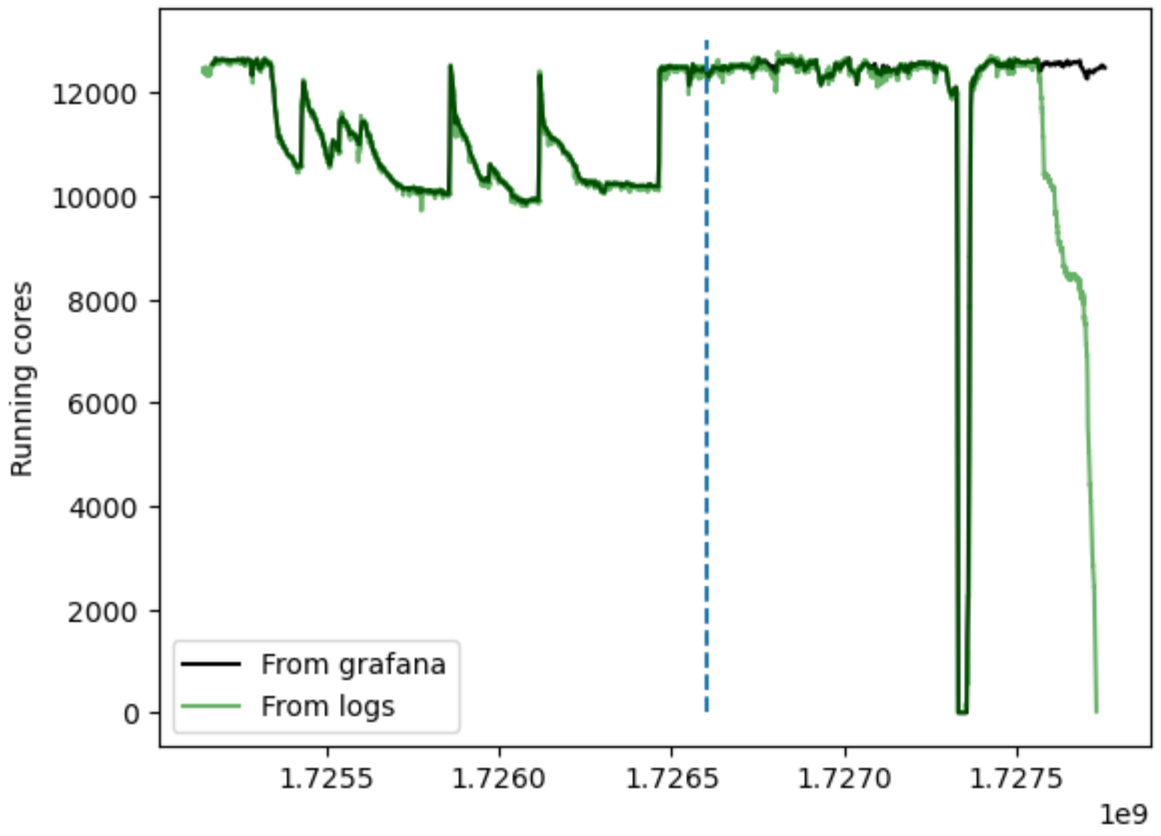
Computation of running cores for September 2024 at PIC

In [10]:
```
t_fi = []
v_fi = []
for ts in np.arange(int(df['enddate'].min()),int(df['enddate'].max()),60)
    t_fi.append(ts)
    v_fi.append(df['request_cpus'][(df['startdate'] <= ts) & (df['enddate'

plt.plot(t_fi, v_fi)
plt.ylabel("Running cores")
plt.show()
```



Compare the estimated running cores with the running cores we monitor in realtime in PIC Grafana for Sept. 2024. At the end of the period, of course, we cannot resemble the farm utilization because there are missing jobs (last 100h).

```
In [11]: time_drop=1726600000

         df_graf = pd.read_csv("./HTCondor_Grafana_all.csv", sep=",", header=0)

         plt.plot(df_graf['Time']/1000, df_graf['All'], label="From grafana", colo
         plt.plot(t_fi, v_fi, label="From logs", color="green", alpha=0.6)
         plt.vlines(time_drop, 0, 13000, linestyle="--")
         plt.ylabel("Running cores")
         plt.legend(loc='best')
         plt.show()
```

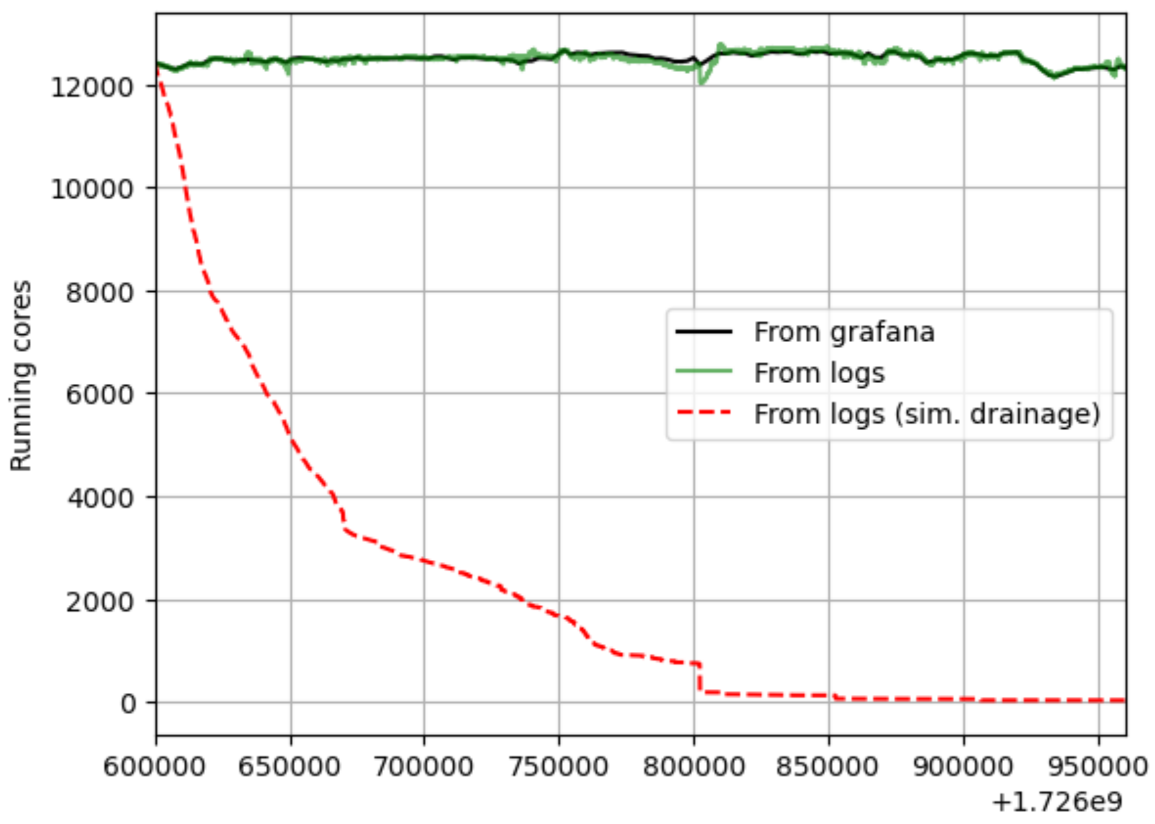


Simulating a natural job drainage

Defining a time in which we stop sending jobs to HTCondor (time_drop), it is easy to simulate a natural job drainage at PIC farm. Simply we need to drop all of the jobs in the dataframe that started after this time_drop (blue dashed line in previous plot).

In [12]:
```
df_drop = df.drop(df[ df['startdate'] > time_drop].index)
```

In [13]:
```
t_fi2 = []
v_fi2 = []
for ts in np.arange(time_drop,time_drop+100*3600,60):
    t_fi2.append(ts)
    v_fi2.append(df_drop['request_cpus'][(df_drop['startdate'] <= ts) & (

plt.plot(df_graf['Time']/1000, df_graf['All'], label="From grafana", colo
plt.plot(t_fi, v_fi, label="From logs", color="green", alpha=0.6)
plt.plot(t_fi2, v_fi2, label="From logs (sim. drainage)", color="red", li
plt.xlim(time_drop,time_drop+100*3600)
plt.ylabel("Running cores")
plt.legend(loc='best')
plt.grid()
plt.show()
```

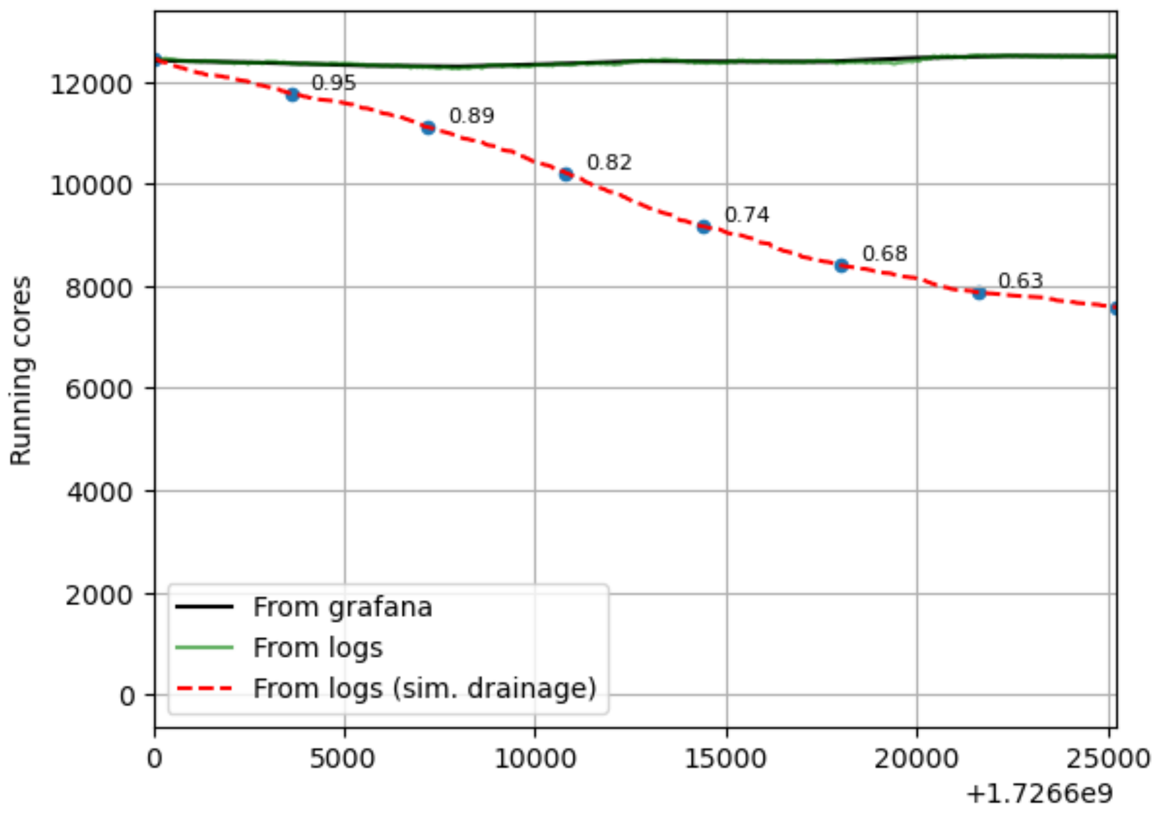


One can see that the farm empties after 100h, which is in agreement to the job duration distribution we made earlier.

In particular, we observe in this simulation that approx. there is a 25% decrease in farm utilization every 4 hours, or approx. 6% every hour.

In [52]:

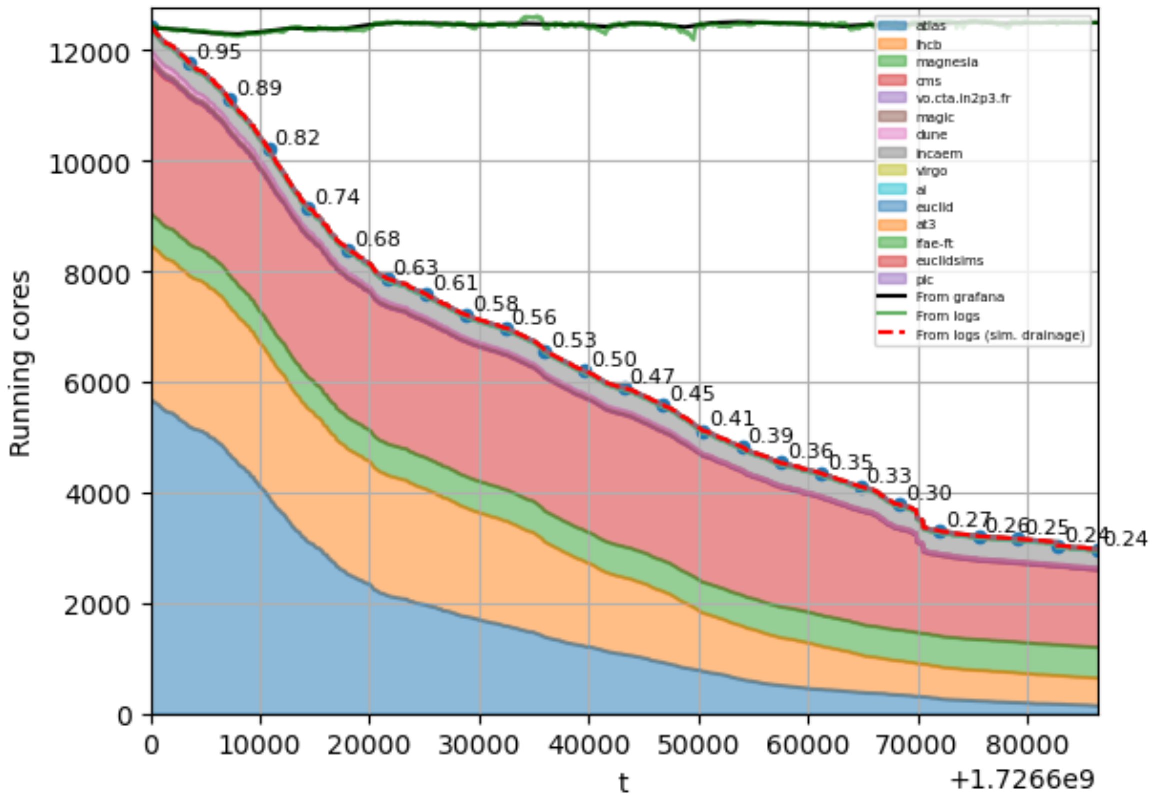
```
plt.plot(df_graf['Time']/1000, df_graf['All'], label="From grafana", colo
plt.plot(t_fi, v_fi, label="From logs", color="green", alpha=0.6)
plt.plot(t_fi2, v_fi2, label="From logs (sim. drainage)", color="red", li

f = []
f_t = []
t_red = []

Nhours = 48

for i in range(0,Nhours+1):
    index=np.where((t_fi2-t_fi2[0]) == 3600*i)[0][0]
    f.append(v_fi2[index])
    f_t.append(v_fi2[index]/v_fi2[0])
    t_red.append(time_drop+i*3600)

plt.scatter(t_red, f, s=20)

Nhours = 6
for i in range(1,Nhours+1):
    plt.text(t_red[i]+500,f[i]+100, "{:.2f}".format(f_t[i]), fontsize=8)

plt.xlim(time_drop,time_drop+(Nhours+1)*3600)
plt.ylabel("Running cores")
plt.legend(loc='best')
plt.grid()
plt.show()
```



Not all of the VOs have the same job duration, hence the queues should be dominated by jobs with longer execution time

In [48]:
```
plotdata = pd.DataFrame()
vo_ord = df_drop['VO'][(df_drop['enddate_wt'] > time_drop)].value_counts(

for vo in vo_ord:
    t_fi_tmp = []
    v_fi_tmp = []
    df_copy = df_drop.copy()
    df_copy = df_copy.drop(df_copy[ df_copy['VO'] != vo].index)
    for ts in np.arange(time_drop,time_drop+100*3600,60):
        t_fi_tmp.append(ts)
        v_fi_tmp.append(df_copy['request_cpus'][(df_copy['startdate'] <=
    plotdata[vo] = v_fi_tmp
    del df_copy
plotdata['t'] = t_fi_tmp
plotdata.set_index('t', inplace=True)
```

In [58]:
```
plotdata.plot.area(alpha=0.5)
plt.plot(df_graf['Time']/1000, df_graf['All'], label="From grafana", colo
plt.plot(t_fi, v_fi, label="From logs", color="green", alpha=0.6)
plt.plot(t_fi2, v_fi2, label="From logs (sim. drainage)", color="red", li

Nhours = 24
plt.scatter(t_red, f, s=20)

for i in range(1,Nhours+1):
    plt.text(t_red[i]+500,f[i]+100, "{:.2f}".format(f_t[i]), fontsize=8)

plt.xlim(time_drop,time_drop+Nhours*3600)
plt.ylim(0,np.max(v_fi))
plt.ylabel("Running cores")
plt.legend(loc='best', fontsize=5)
plt.grid()
plt.show()
```



Next steps

Conduct extended simulations to observe drainage patterns across varying load conditions

Analyze the impact of job types and VO-specific job durations on natural drainage: how different workloads contribute to natural drainage cycles

Estimate power consumption before and after drainage (ipmitool) - we know which job slots are getting free, and in which machine

Evaluate potential carbon emission reductions in these drainage scenarios

Develop machine learning models for predictive power scaling

Design a feedback loop to HTCondor for real-time power modulation based on green energy cycles