RADICAL-Analytics

Matteo Turilli, Andre Merzky, Alessio Angius, Shantenu Jha

Rutgers University

Outline

- Specificities of analytics for scientific middleware.
- State and data models for analytics back ends.
- Benefits and Challenges a model-based approach to analytics back ends.
- Case study: RADICAL-Pilot and RADICAL-Analytics.
- RADICAL-Analytics workflow.
- Examples.
- Conclusions.

Analytics for Scientific Middleware

- **Goal:**
  - enabling statistical analysis of runtime data produced by scientific middleware.
- **Objectives:**
  - Developing state and data models to support analytics independent from the specifics of resources and middleware implementations;
  - Distinguishing between analytics about the middleware behavior and analytics about the workload execution.
  - Separating back-end and front-end for analytics to decouple data collection, wrangling, filtering, analysis, and plotting stages.
- **Challenges:**
  - arbitrary analytical methodologies;
  - arbitrary type and number of resources;
  - arbitrary middleware design and development;
  - arbitrary data models and collection mechanisms.

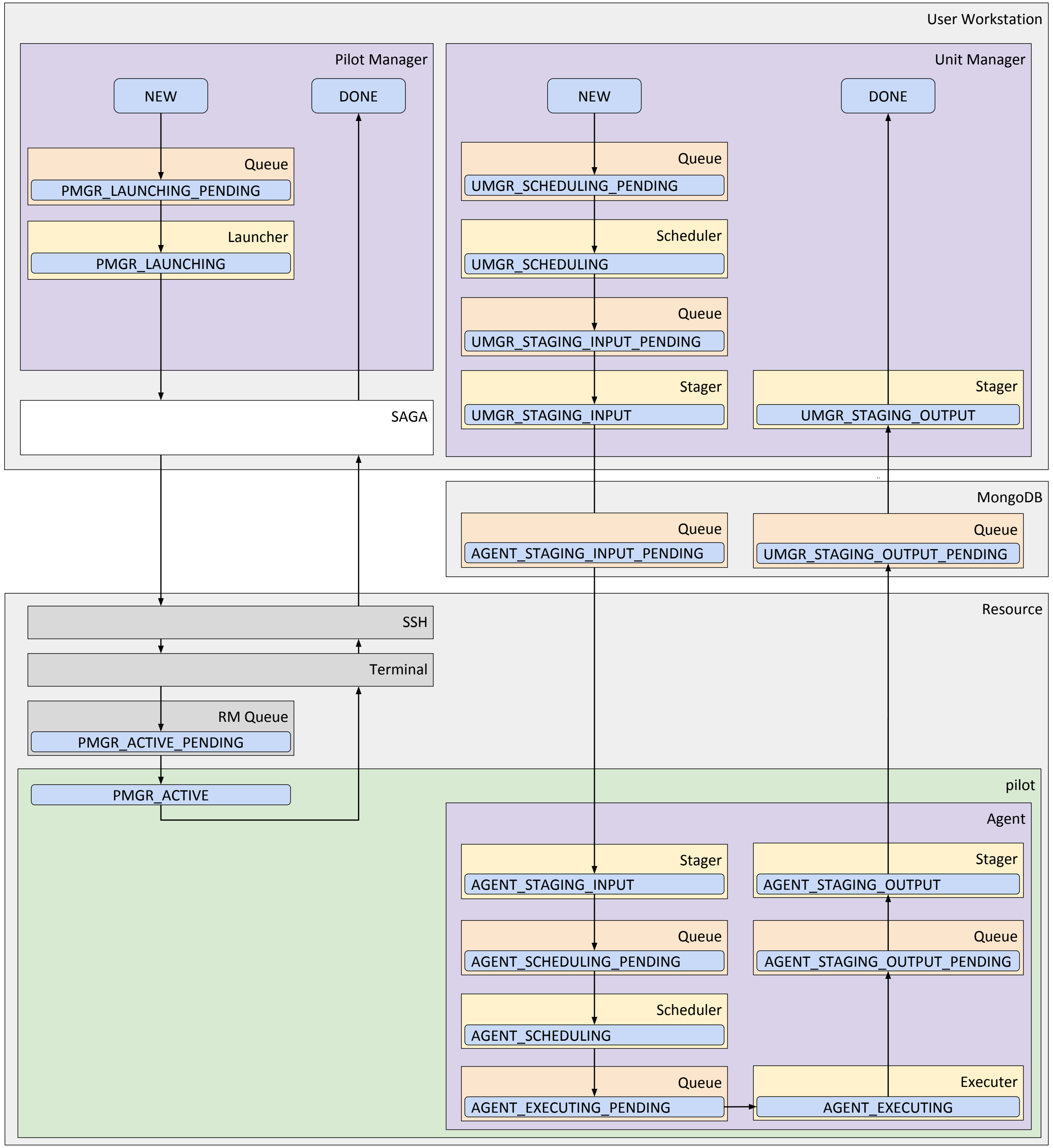
State and Data Models

- **Element**: functional unit of the middleware code. E.g., functions, methods.
- **Event**: moment in time recorded by an element. E.g., bootstrap, write output.
- **Entity**: logical unit of the middleware or of the workload. E.g., manager, agent, task, file.
- **State**: period of time delimited by two events, i.e., transitions. E.g., queuing, executing, staging.

- State model: a sequence of states, assuming sequences of atomic state transitions.
- Data model: events and transitions as recorded by a middleware implementation while executing a given workload.
- Each transition is performed by a specific entity of the middleware on a specific entity of the workload. Transitions of a state and of a state model are assumed to be ordered as a time series. E.g., <T0-S1t0, T1-S1t1, T2-S2t0, T3-S2t1>.
- Details like how to record and store events and transitions, the time stamp precision, or the interfaces to access the records are implementation specific.

RADICAL-Pilot State Model

- Simplified: each state can transition to the Failed and Cancel final states.
- Gray boxes: physical locations where the code is executed.
- Purple boxes: Entities of RADICAL-Pilot.
- White boxes: Third-party software components.
- Dark gray boxes: Resource software components.
- Orange boxes: queues.
- Blue boxes: state transitions.
- Green box: pilots.



RADICAL-Analytics

- **API:**
  - session.describe()
  - session.list()
  - session.get()
  - session.filter()
  - session.ranges()
  - session.duration()
  - session.concurrency()
  - session.consistency()
  - session.accuracy()

- **Implementation:**
  - Coded as a stand-alone Python module.
  - Currently, it embeds the specification of the RADICAL-Pilot state model but it has been designed to work with arbitrary state models.
  - Designed to be extensible: offers a minimal but not complete set of methods.
  - Designed to be as transparent as possible to the experiment analysis. No higher order methods are offered for run aggregation, comparison or plotting.
  - GitHub repository: https://github.com/radical-cybertools/radical.analytics
  - Documentation: https://readthedocs.org/projects/radicalanalytics/

Example: Analytics Workflow

1. Raw data:

```
#time,name,uid,state,event,msg
1484625129.3653,umgr.0000:,,,sync abs,radical:144.76.72.175:1484625129.36:1484625129.36:ntp
1484625129.3655,umgr.0000:MainThread,umgr.0000,,initialize,
1484625129.3742,umgr.0000:umgr.0000.idler._profile_flush_cb,,,flush,
1484625129.4034,umgr.0000:MainThread,umgr.0000,,create umgr,
1484625131.4599,umgr.0000:MainThread,,,UMGR setup done,
1484625131.4759,umgr.0000:MainThread,unit.000000,NEW,advance,
1484625131.4763,umgr.0000:MainThread,unit.000001,NEW,advance,
1484625131.4768,umgr.0000:MainThread,unit.000002,NEW,advance,
```


```
"bulk_collection_size": 100,
"bulk_collection_time": 1.0,
"cname": "PMGRLaunchingComponent",
"components": {
    "UpdateWorker": 1
},
"cores": 1,
"cores_per_node": 0,
"db_poll_sleeptime": 1.0,
"dburl": "mongodb://matteo:matteo@ds035385.mlab.com:35385/aimes-aws",
"debug": 10,
"global_sandbox": "/home/mturilli/radical.pilot.sandbox",
"heartbeat_interval": 100,
"heartbeat_timeout": 3000,
"logdir": ".",
"lrms": "FORK",
"max_io_loglength": 1024,
"mpi_launch_method": "",
"network_interface": "lo",
"owner": "agent_0",
"pilot_id": "pilot.0005",
```


2. Wrangled data:

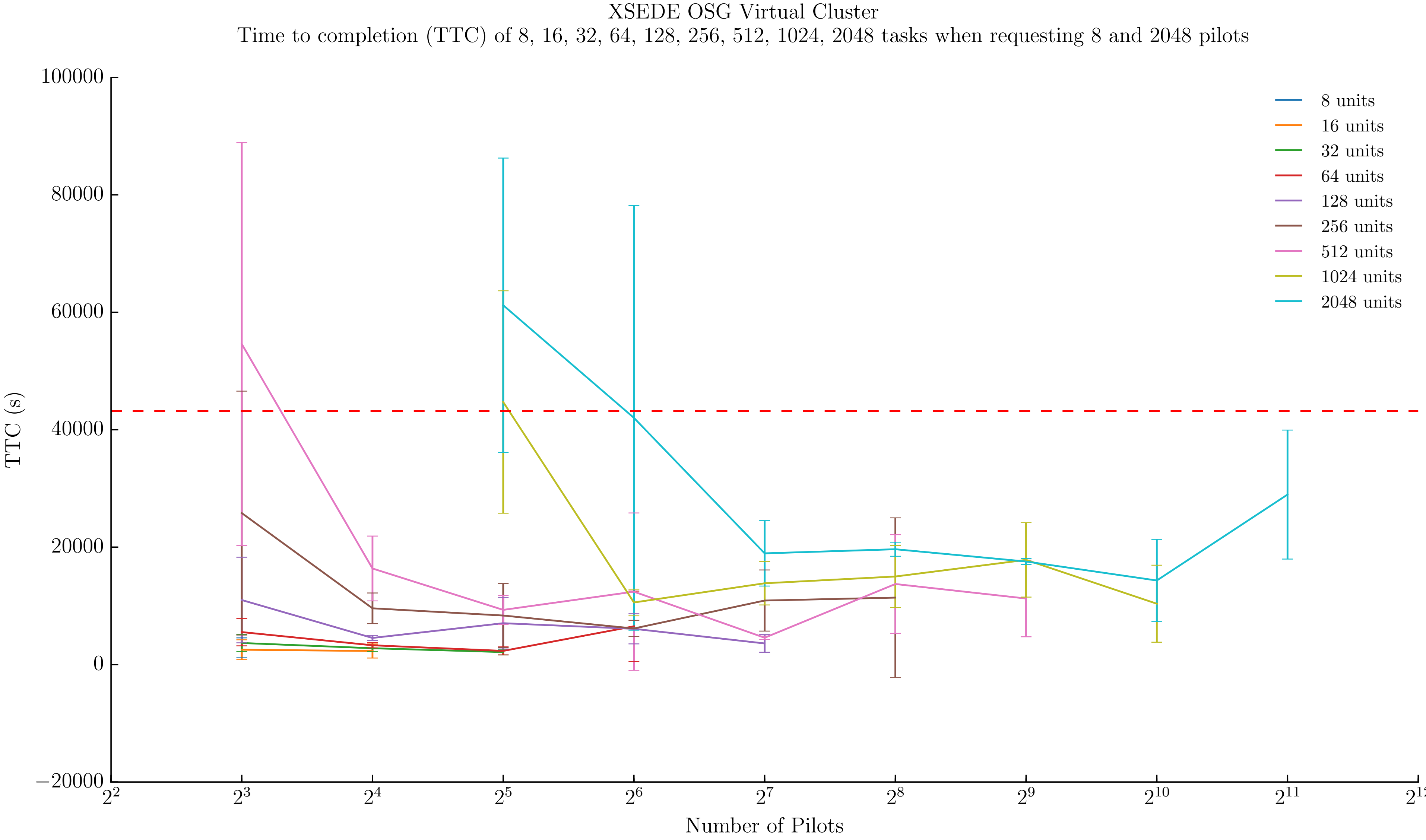
```
,P_LRMS_QUEUING,P_LRMS_RUNNING,P_LRMS_SUBMITTING,P_PMGR_QUEUING,P_PMGR_SCHEDULING,experiment,hid,nunit,pid,sid
0,448.921200037,1096.9770999,40.8440999985,0.00209999084473,0.111500024796,exp1,b7c6cf8a3ae1.172.19.0.3,3,pilot.0000,rp.session.radical.mingtha.017033.0007
1,274.998300076,1274.11059999,40.8440999985,0.00209999084473,0.111500024796,exp1,phys.uconn.edu,4,pilot.0001,rp.session.radical.mingtha.017033.0007
2,217.17930007,1330.81699991,40.8440999985,0.00209999084473,0.111500024796,exp1,ucsd.edu,4,pilot.0002,rp.session.radical.mingtha.017033.0007
3,220.585400105,1321.01849985,40.8440999985,0.00209999084473,0.111500024796,exp1,ucsd.edu,5,pilot.0003,rp.session.radical.mingtha.017033.0007
```

3. Analysis: load csv file(s) into analytics front end. E.g., R, Pandas, SPARK, Stata, SPSS and so on.

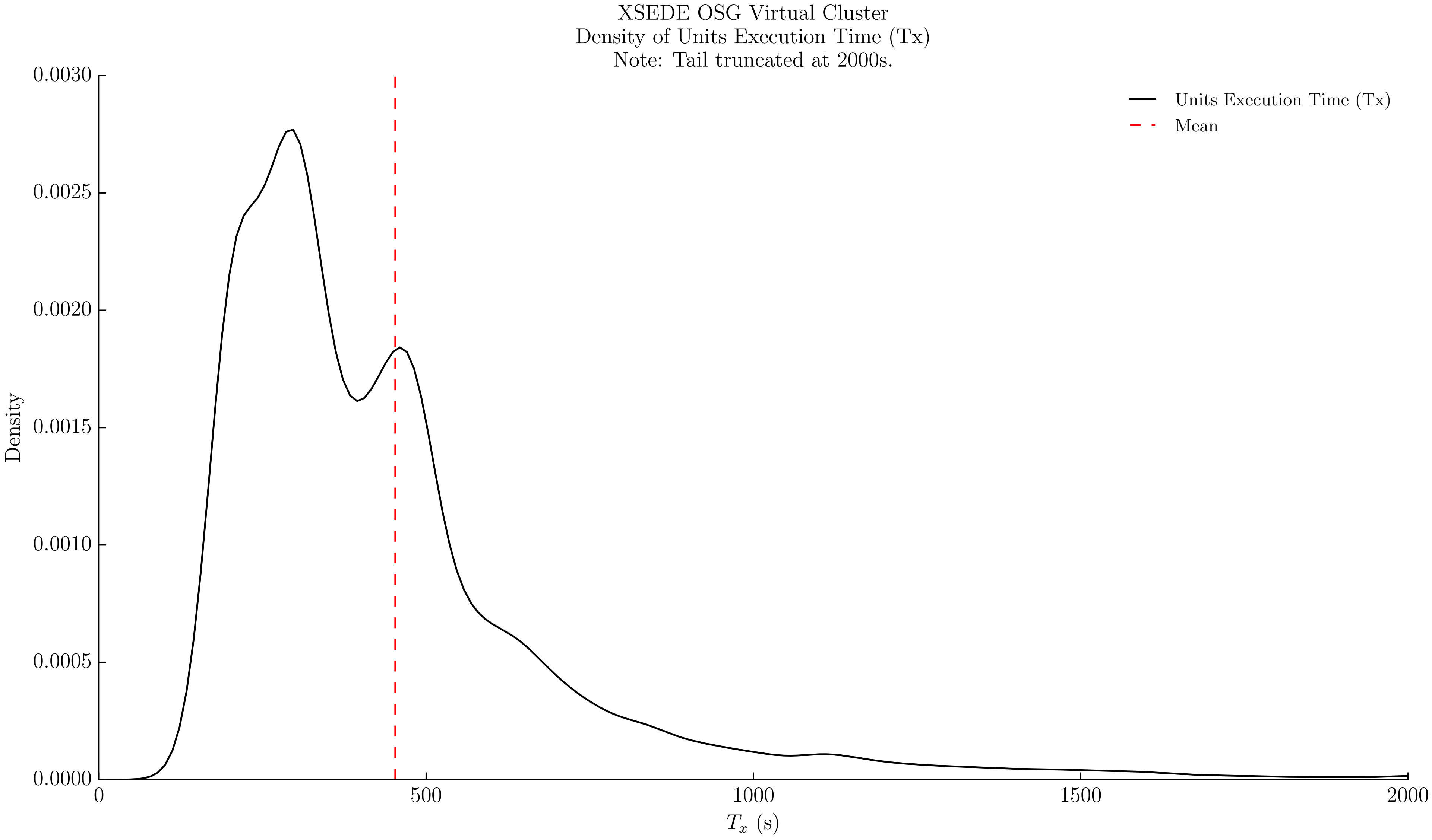
Load wrangled data saved in .csv files.

In [15]:
```
sessions = pd.read_csv('data/sessions.csv', index_col=0)
pilots = pd.read_csv('data/pilots.csv', index_col=0)
units = pd.read_csv('data/units.csv', index_col=0)
```

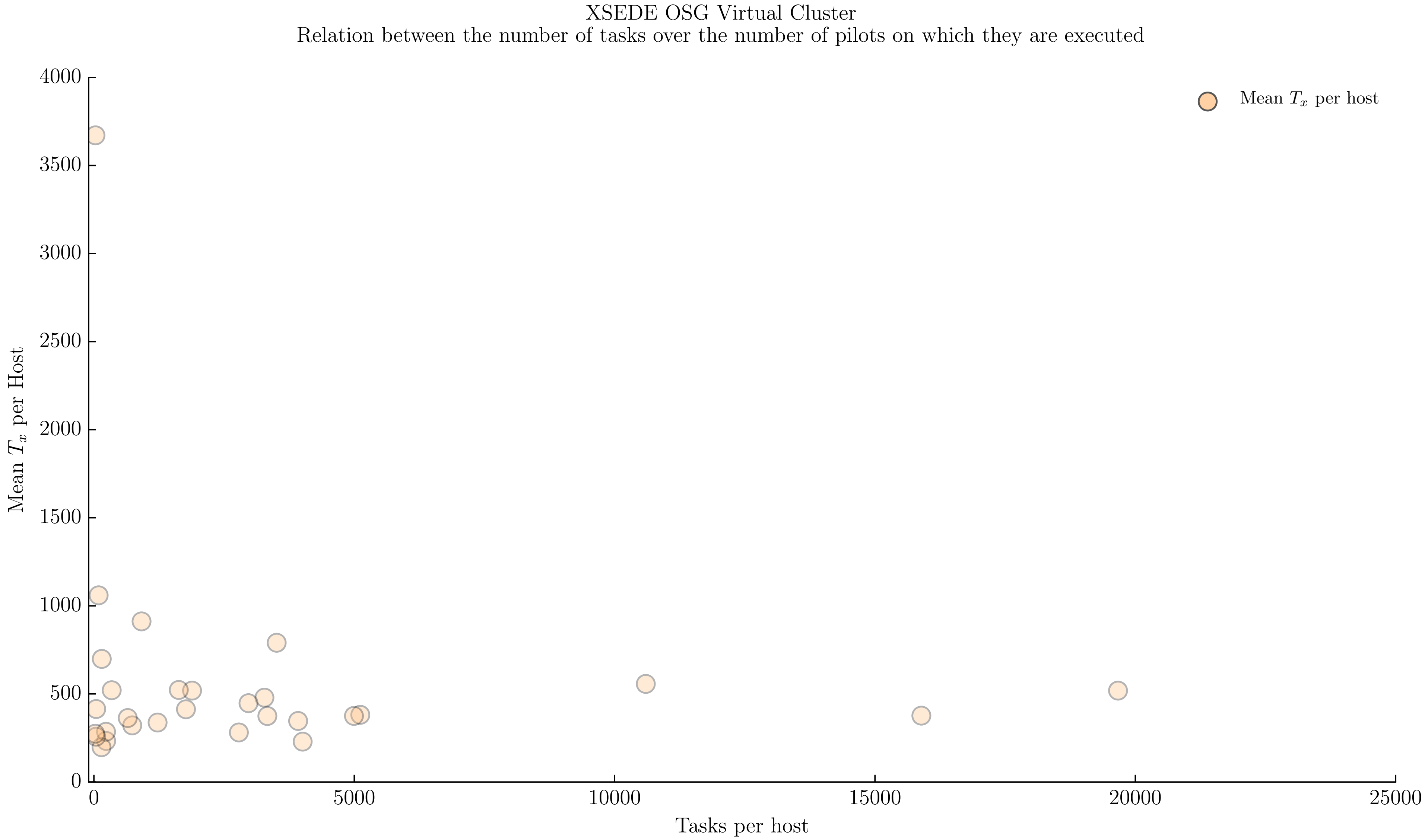
XSEDE OSG: Time To Completion (TTC)



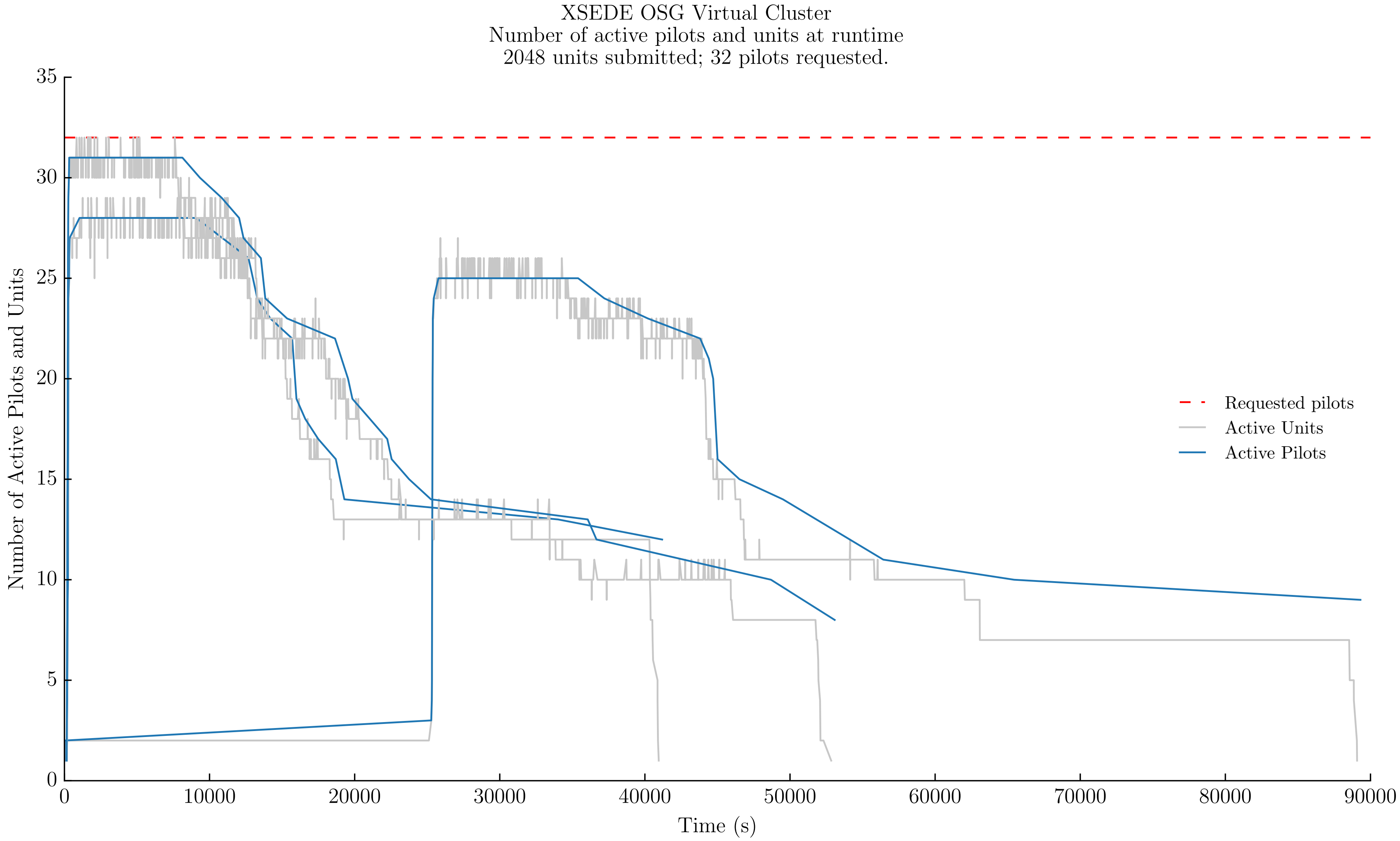
XSEDE OSG: Distribution Task Execution Time (Tx)



Example XSEDE OSG: Non-Correlation Tx and #Hosts



XSEDE OSG: Concurrency Pilots and Tasks



Benefits and Challenges

- **Benefits:**
  - Entities modeled as logical units enable analytics for both middleware and workload.
  - Explicit definition of the state models enables the automation of consistency tests. This works both for middleware implementation and workload execution.
  - Uniform interpretation across publications enables data comparison and reuse across multiple lines of research.
  - Avoid endless duplication of scripted solutions for specific point analyses.

- **Challenges:**
  - Design-heavy development methodology leads to slower prototyping but (hopefully) more robust production implementations.
  - Automatic derivation of state and data models from code. Models inferred from the code base to avoid the progressive divergence between implementation and specification.
  - Accounting for concurrency: events, transitions and states can be recorded by and for multiple entities.

Conclusions

- Defining explicit state and data models.
- Decoupling backend and frontend for analytics.
- Supporting analytics about the middleware and about the execution of the workload.
- Automating consistency and accuracy tests.
- Uniform data model across diverse analyses.
- Accounting for concurrency of multiple timelines, one for each stateful entity.
- Agnostic towards type and design of the middleware used to produce raw data.
- Agnostic towards analysis models and tools.
- Agnostic towards library or service oriented frontends, i.e., R vs Spark.