8th International Conference "Distributed Computing and Grid-technologies in Science and Education" (GRID 2018, Dubna, 10-14 September 2018)

Supporting Efficient Execution of Many-Task Applications with Everest

Oleg Sukhoroslov

Institute for Information Transmission Problems
of the Russian Academy of Sciences (Kharkevich Institute)

10.09.2018

Many-Task Applications

- Loosely-coupled applications consisting of a (large) number of computational tasks which can be executed (more or less) independently
- Bag-of-tasks applications
  - No data/control dependencies between tasks
  - Parameter sweeps, Monte Carlo simulations, image rendering
- Cooperative problem solving
  - Exchange of information between tasks
  - Branch-and-bound method
- Workflows
  - Multiple tasks with control or data dependencies (DAG)
  - Automation of computational and data processing pipelines

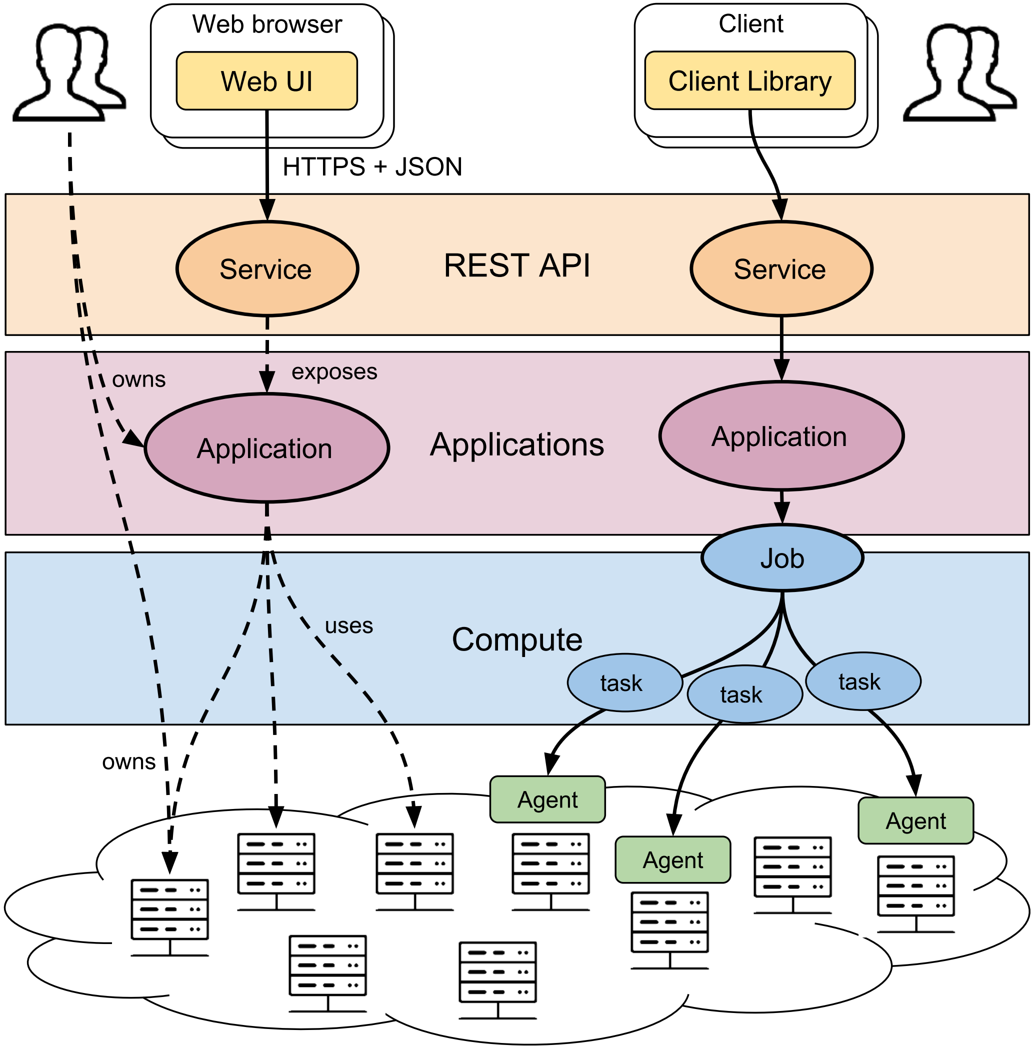
Challenges

- Management of a large number of tasks
- Accounting for dependencies between tasks
- Coordination and data exchange between tasks
- Execution on multiple distributed resources
- Task scheduling
- Accounting for local resource policies
- Dealing with failures

Everest

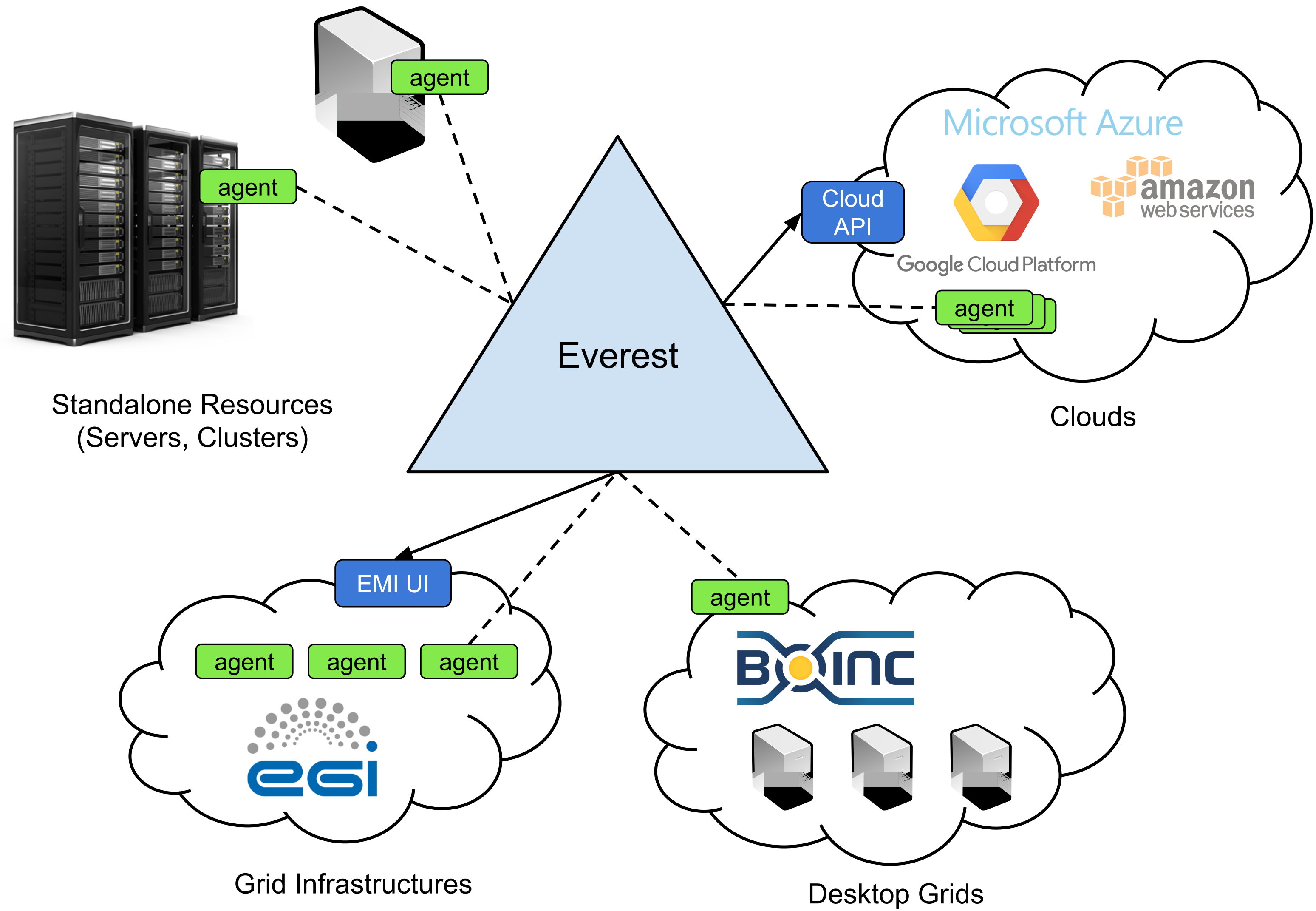
- Web-based platform supporting
  - Publication of computational applications as services
  - Execution of applications on external computing resources
  - Sharing applications and resources with other users
  - Composition of applications (workflows)
- Platform as a Service
  - Remote access via web browser and REST API
  - Single platform instance can be accessed by many users
  - No installation is required
- Public instance with open registration
  - http://everest.distcomp.org/

Everest



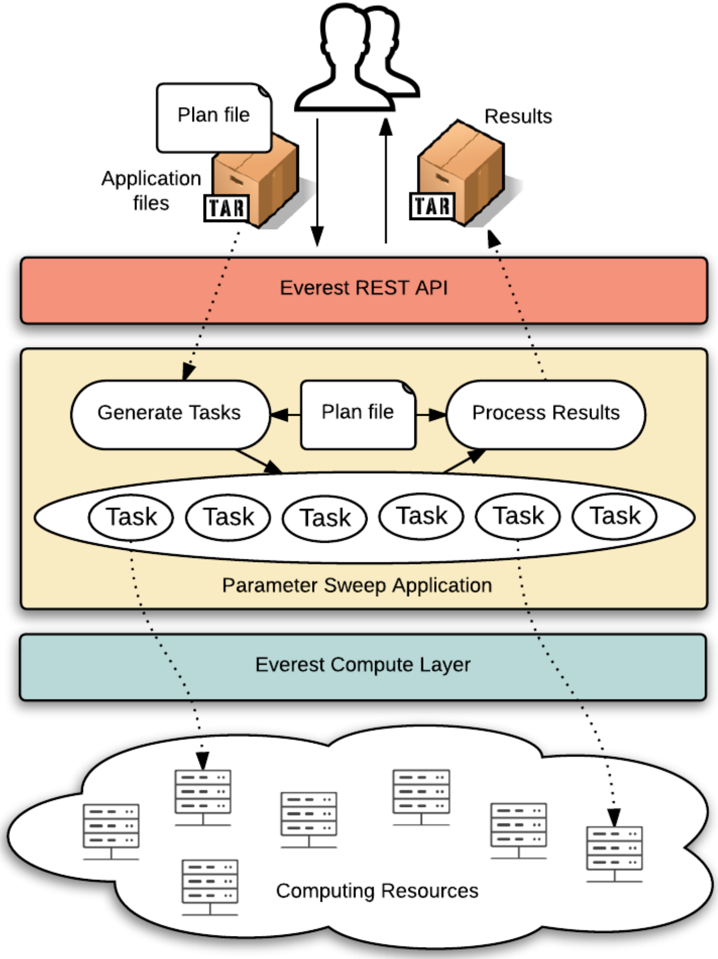
Web browser
Web UI
Client
Client Library
HTTPS + JSON
Service
REST API
Service
owns
exposes
Application
Applications
Application
Job
uses
Compute
task
task
task
owns
Agent
Agent
Agent

Computing Resurces



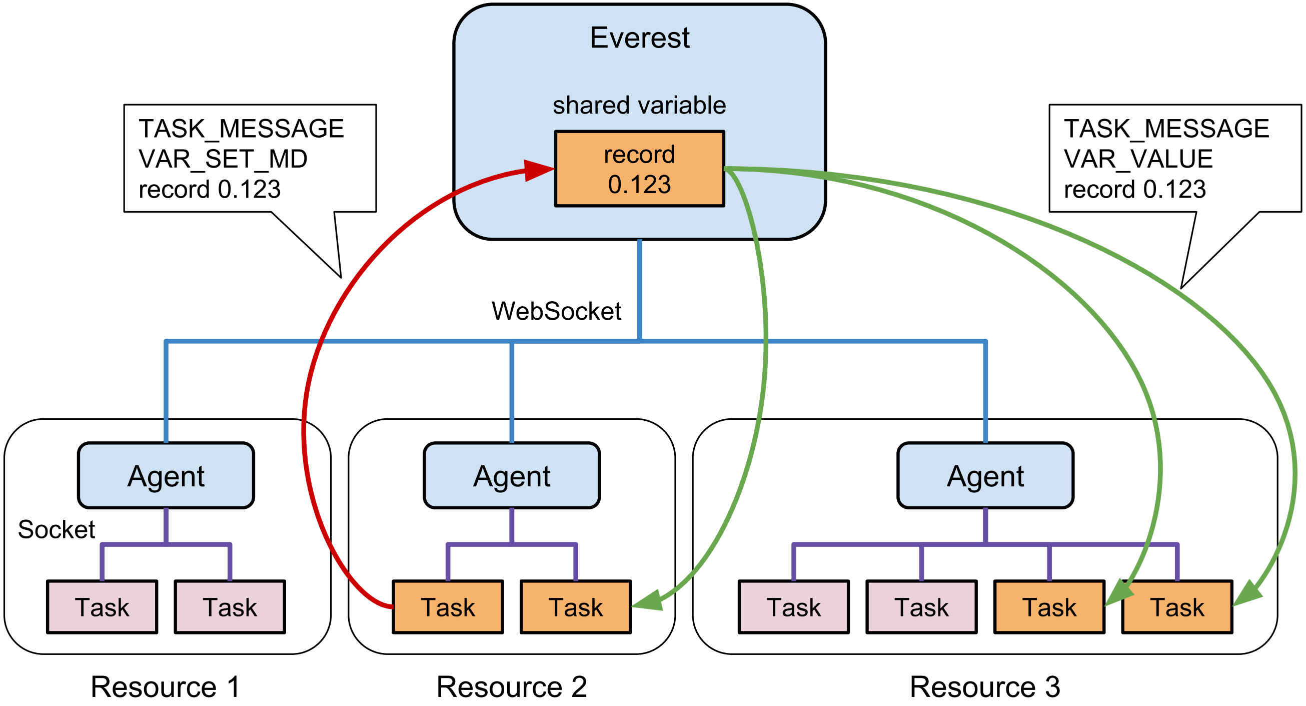
ParameterSweep Service

```
parameter n from 1 to 100 step 1

input_files @run.sh vina
write_score.py protein.pdbqt
input_files ligand${n}.pdbqt
config.txt

command ./run.sh

output_files ligand${n}_out.pdbqt
log.txt @score

criterion min $affinity
```



Messaging / Shared Variables

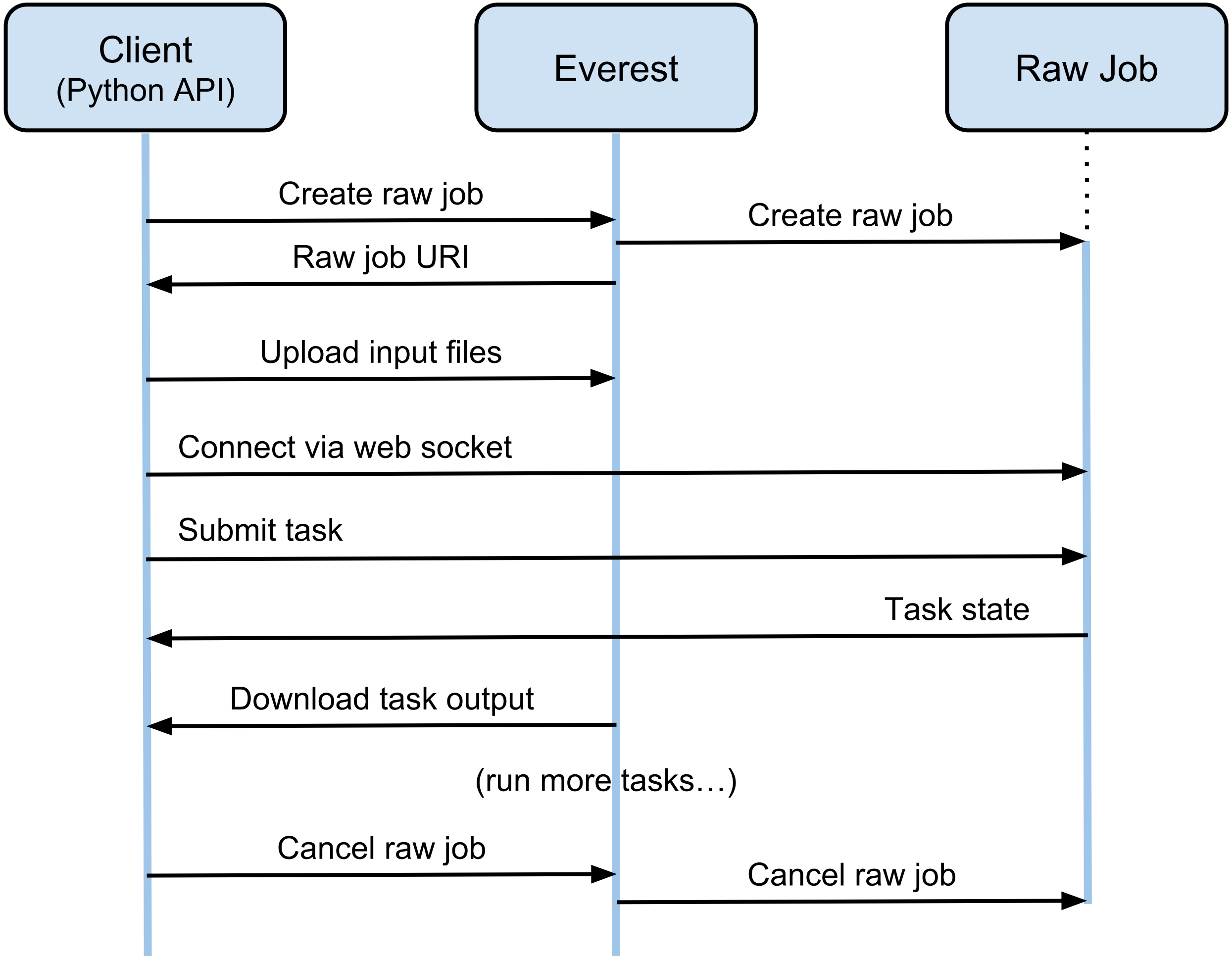
Motivation: exchange of incumbent values (records) in parallel branch-and-bound method



Voloshinov V., Smirnov S., Sukhoroslov O. Implementation and Use of Coarse-grained Parallel Branch-and-bound in Everest Distributed Environment // Procedia Computer Science. Volume 108, 2017.

Raw Job

A low-level interface for flexible management of tasks



Accounting for Local Resource Policies

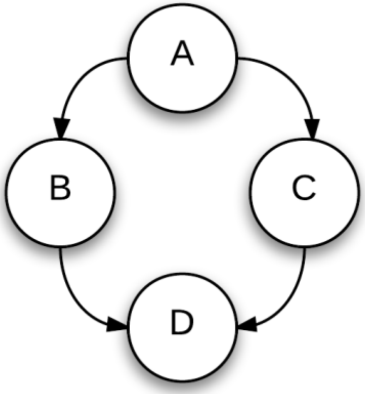
- Running application on a HPC cluster
  - Single cluster job per Everest task by default
- Limit on the maximum number of jobs per user imposed by the cluster administrators
  - May not allow to fully utilize the resource
- Solution
  - Pack multiple tasks in a single cluster job
  - Advanced adapter for Slurm which supports submission of complex jobs consisting of multiple tasks

Dealing with Task Failures

- Distinguish between critical and recoverable errors
  - Non-zero exit code
  - Agent is disconnected, data download/upload error
- Automatically retry the task after a recoverable error
  - Up to 3 times by default
- Allow application developer to enable retry after a critical error
- Account for temporary failures
  - Network failures are common, agent is temporarily disconnected
  - Do not reschedule tasks immediately to avoid wasting compute time

Workflows with Python API

```
import everest

session = everest.Session(
  'https://everest.distcomp.org', token = '...'
)

appA = everest.App('52b1d2d13b...', session)
appB = everest.App('...', session)
appC = everest.App('...', session)
appD = everest.App('...', session)

jobA = appA.run({'a': '...'})
jobB = appB.run({'b': jobA.output('out1')})
jobC = appC.run({'c': jobA.output('out2')})
jobD = appD.run({'d1': jobB.output('out'), 'd2': jobC.output('out')})

print(jobD.result())

session.close()
```



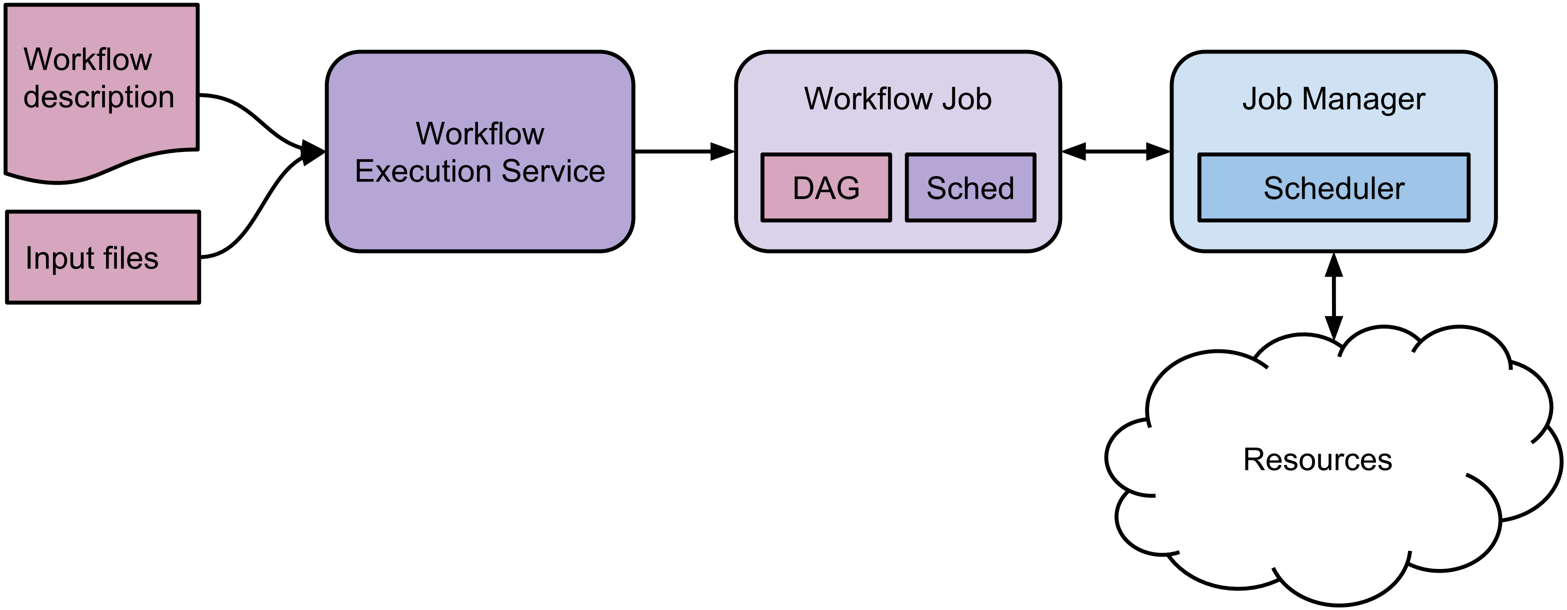
Limitations

- Dependencies between tasks are managed externally by a user application
- Only the tasks ready to run are (dynamically) submitted to Everest
- The platform is not aware of complete DAG
- Not possible to use advanced workflow scheduling algorithms

Workflow Execution Service (WIP)

A general-purpose service for execution of workflows

- Workflow description is passed in YAML format
- Workflow job incapsulates corresponding DAG and internal scheduler
- Everest runs tasks selected by the internal scheduler



Task Scheduling

- Two-level scheduling mechanism that allows to plug-in different application-level scheduling algorithms
  - Level 1: fair distribution of available resources among the jobs
  - Level 2: offering resources to a job, the job selects a task to run using the application-level scheduler
- Scheduler implementations
  - Sukhoroslov O. V., Nazarenko A., Aleksandrov R. An experimental study of scheduling algorithms for many-task applications // The Journal of Supercomputing, 2018
  - ParameterSweep (bag-of-tasks): OLB, MaxMin
  - Workflows: OLB, DLS
- Estimation of task execution and data transfer times
  - Statistics from previous executions

Conclusion

- Everest implements ready-to-use tools for automated execution of many-task applications in distrubuted environments
  - Integration with different resource types
  - Ability to use multiple resources for running an application
  - ParameterSweep service
  - Task messaging / shared variables
  - Raw Job interface
  - Workflows via Python API
- Ongoing work on the new functionality and improvements
  - Workflow execution service
  - Application-level schedulers