RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

An Introduction to SAGA and BigJob

http://radical.rutgers.edu

OSDC-PIRE Workshop

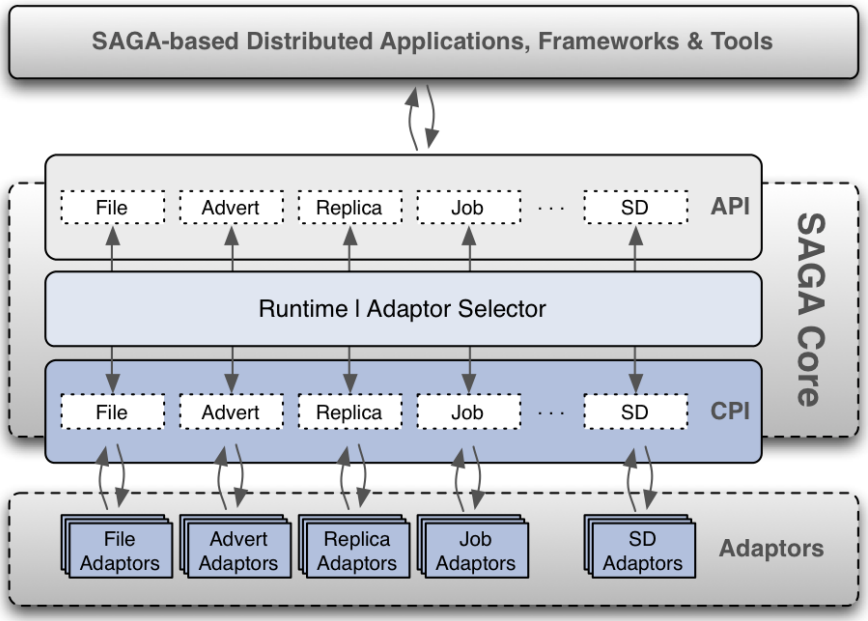
Edinburgh, 20 June 2013

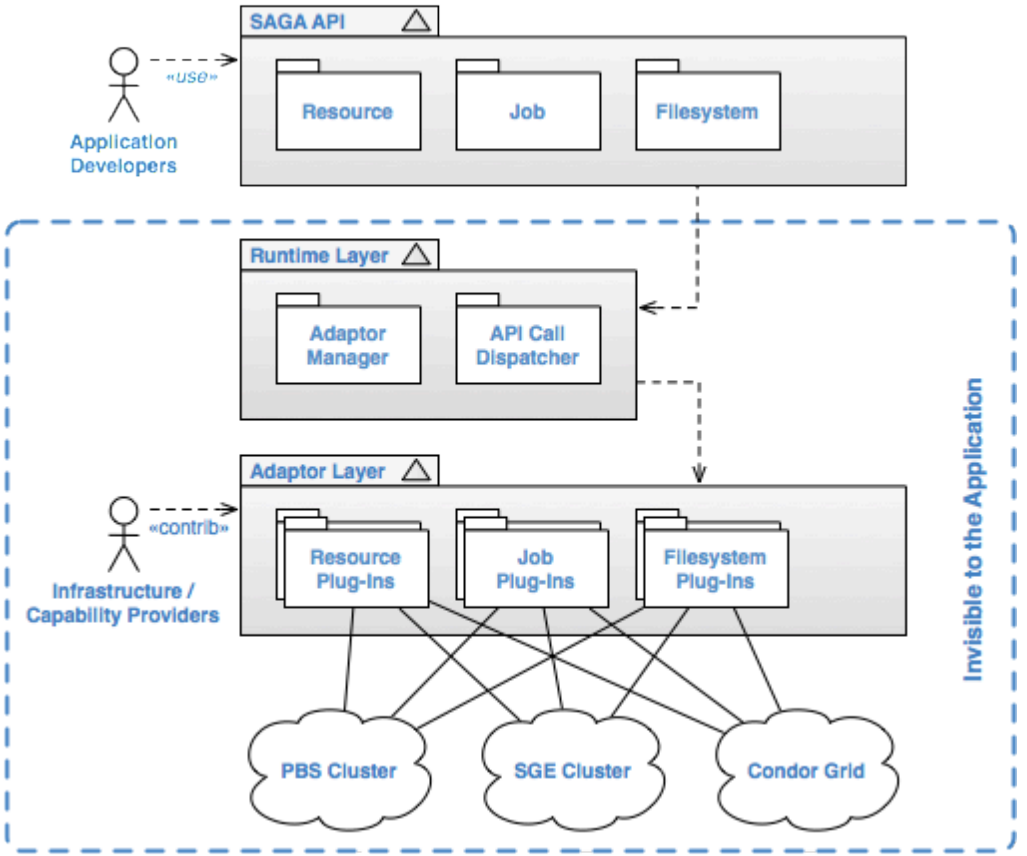
Overview

- SAGA: Simple API for Distributed (“Grid”) Applications
  - Native Python implementation of Open Grid Forum GFD.90
  - Allows access to different middleware / services through unified interface
  - Provides access via different backend plug-ins (“adaptors”)

- Unified Semantics:
  - saga-python provides not only a common API, but also unified semantics across heterogeneous middleware:
    - Transparent Remote operations (SSH / GSISSH tunneling)
    - Asynchronous operations
    - Callbacks
    - Error Handling

SAGA: Standard for Distributed Applications





Supported Middleware and Services

- Job Submission Systems
  - SSH, GSISSH, Condor, Condor-G, PBS(-Pro), TORQUE, SGE, SLURM
- File / Data Management
  - SFTP, GSIFTP, HTTP, HTTPS, (iRODS under development)
- Resource Management / Clouds
  - Amazon EC2 (‘libcloud’-based)

Application examples

- Application is a broad term: “one person’s application is another person’s tool (building block)”.
- saga-python is used in a plethora of contexts:
  - DCI federation: Cloud v.s. Grid v.s. HPC (or hybrid?)
  - Pilot job and Master-Worker frameworks
  - Science gateways and web portals
  - Custom, domain-specific distributed applications

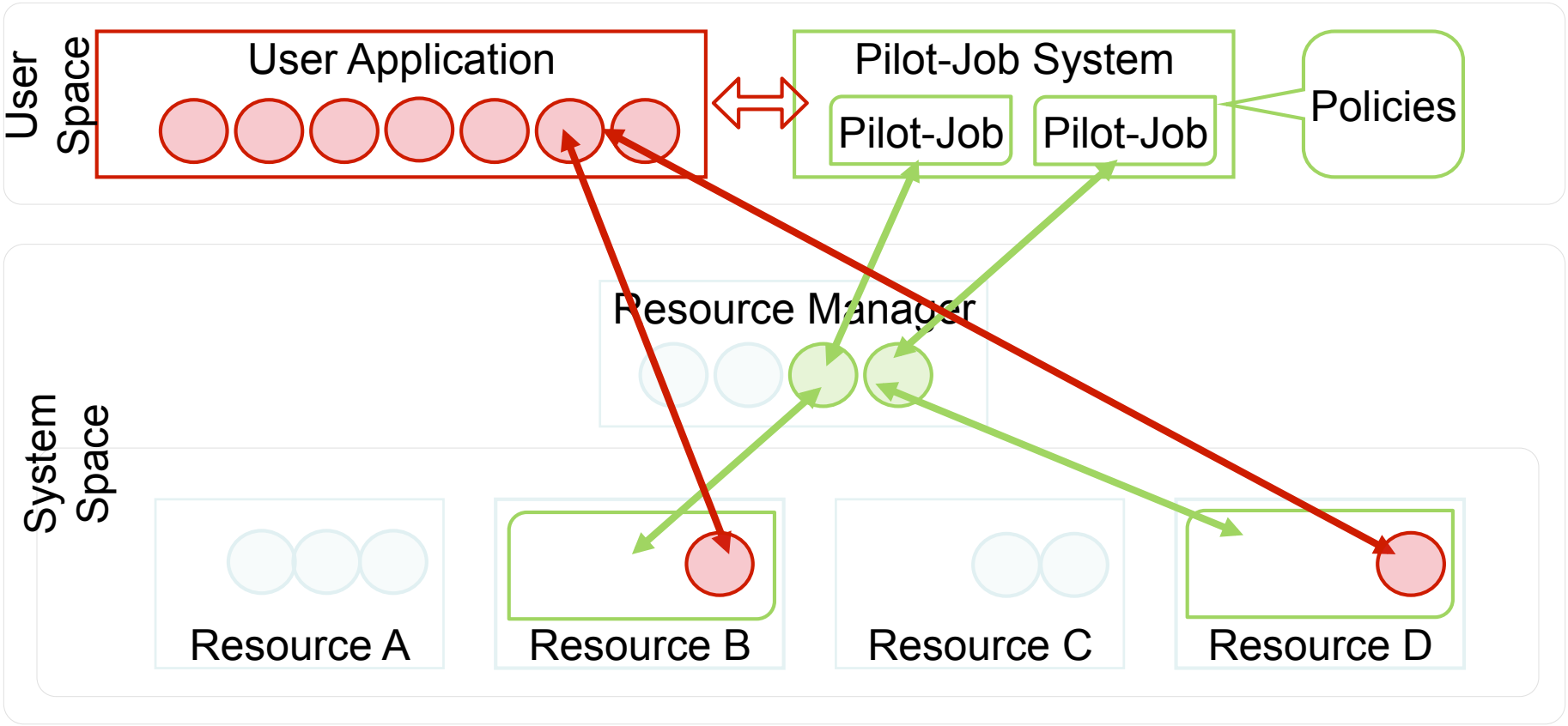
Getting Started

- saga-python website: http://saga-project.github.io/saga-python/
- twitter: https://twitter.com/SAGAGridAPI
- mailing-list: https://groups.google.com/forum/?fromgroups#!forum/saga-users

Introduction to Pilot-Jobs

Introduction to Pilot Jobs

- Working definition: a system that generalizes a placeholder job to provide multi-level scheduling to allow application-level control over the system scheduler via a scheduling overlay



Introduction to Pilot-Jobs (2)

- Working definitions:
  - A system that generalizes a placeholder job to provide multi-level scheduling to allow application-level control over the system scheduler via a scheduling overlay
  - "*.. defined as an abstraction that generalizes the reoccurring concept of utilizing a placeholder job as a container for a set of compute tasks; an instance of that placeholder job is referred to as Pilot-Job or pilot.*"

- Advantages of Pilot-Job systems:
  - Avoid limitations of system-level only scheduling
    - Application Level Scheduling: Abstraction between application and resource layer
  - Flexible Resource Management
    - Enable the "slicing and dicing" of resources
  - Move control and flexibility "upwards"
    - e.g., finer grained temporal control

PilotJob Paradigm

Based upon analysis of Pilot-Job several implementations

Architecture: Three distinct logical elements:

- **Workload Manager**: Responsible for making available the tasks to the executor alongside the needed data and retrieving results
- **Task Executor**: Responsible for executing the tasks while managing their data.
- **Communication and Coordination** (C-C): Patterns allow for and regulate the interaction between (and within) these two components.

Execution Patterns: Based on multi-level scheduling and late-binding:

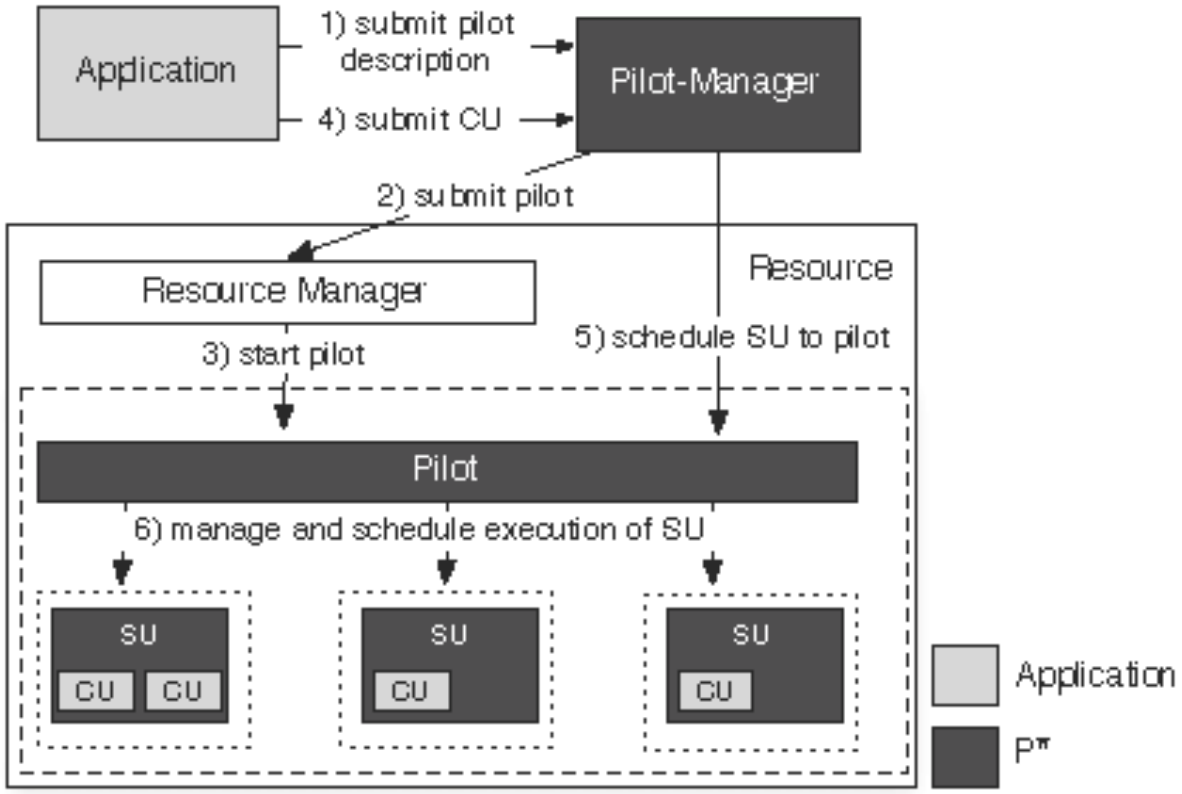
- **Multi-level scheduling**. Tasks of a workload are scheduled on one or more pilots and the pilots are then scheduled on a given resource

Capability/Functionality: A system that generalizes a placeholder job to provide multi-level scheduling to allow application-level control over the system scheduler via a scheduling overlay

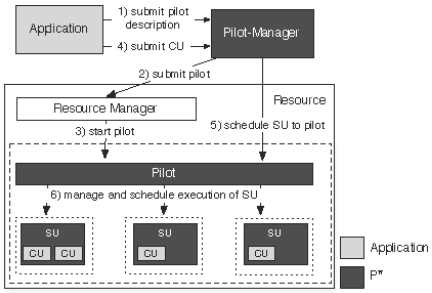
P*: Theory and Practice of Pilot-Jobs

P* Model: Elements, Characteristics and API

- **Elements:**
  - Pilot-Compute (PC)
  - Pilot-Data (PD)
  - Compute Unit (CU)
  - Data Unit (DU)
  - Scheduling Unit (SU)
  - Pilot-Manager (PM)
- **Characteristics:**
  - Coordination
  - Communication
  - Scheduling
- **Pilot-API**



P* Elements



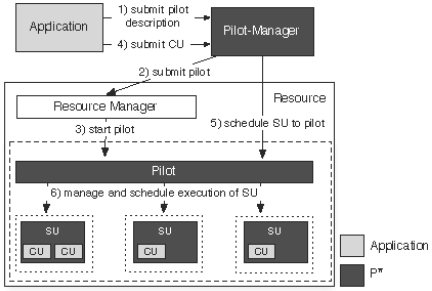
Pilot-Compute

- The placeholder entity that gets submitted to a resource
- Also, associated with the role of an agent:
  - collects information
  - manages the resources allocated
  - exchanges data
- Executes application code

Pilot-Data

- The placeholder entity that represents a storage resource (reservation)
- Can have the role of an agent:
  - collects information
  - manages the resources allocated
- Physically stores the data

P* Elements



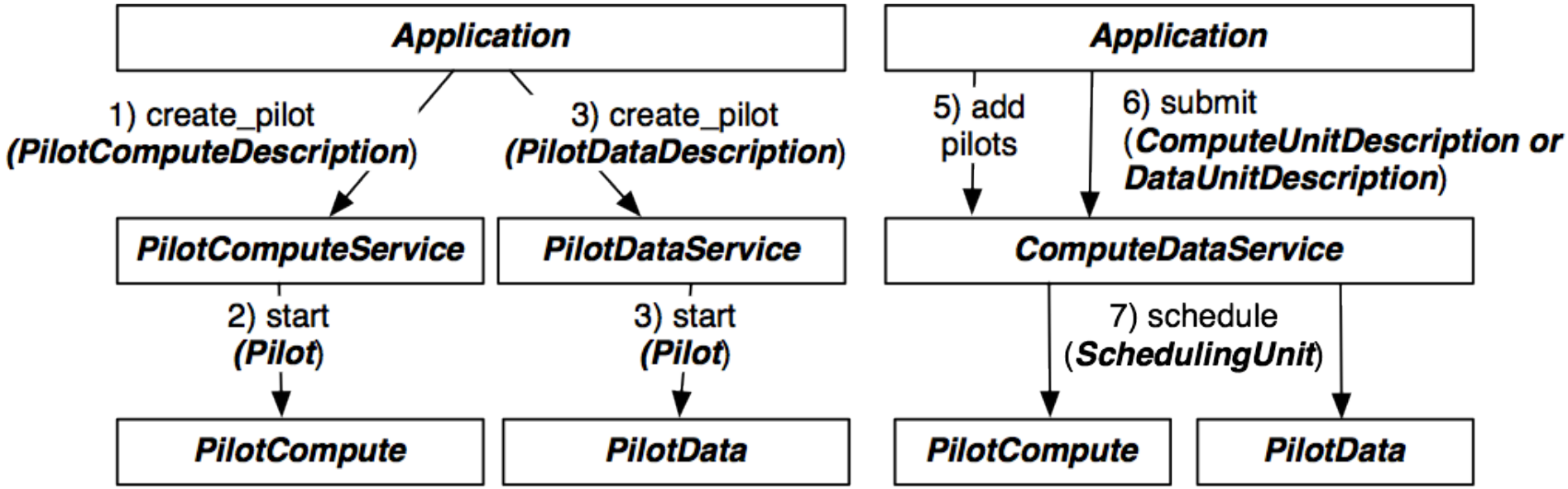
Compute Unit (CU)

- Is defined by the application
- Encapsulates a self-contained piece of work that is submitted to the PJ system
- E.g.:
  - task, job, rpc, web service call, etc.

Data Unit (DU)

- Is defined by the application
- Encapsulates a self-contained piece of logical data that is submitted to the PJ framework
- E.g.:
  - file, chunk, database, etc.

Pilot-API: Unified API to Pilot-Compute and Pilot-Data



Managing Pilots

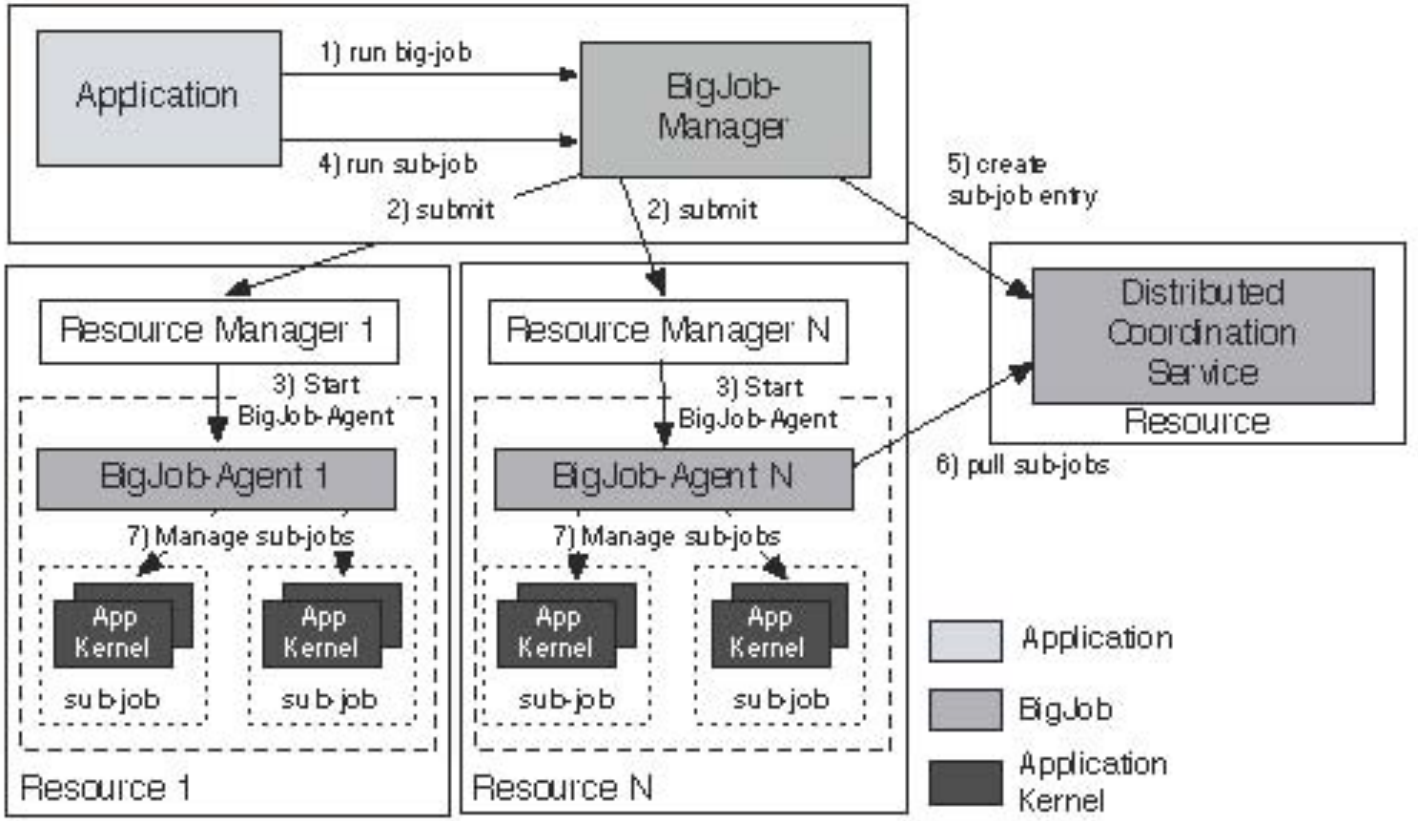
Managing Application Workload

BigJob: A Reference Implementation of the P* Model

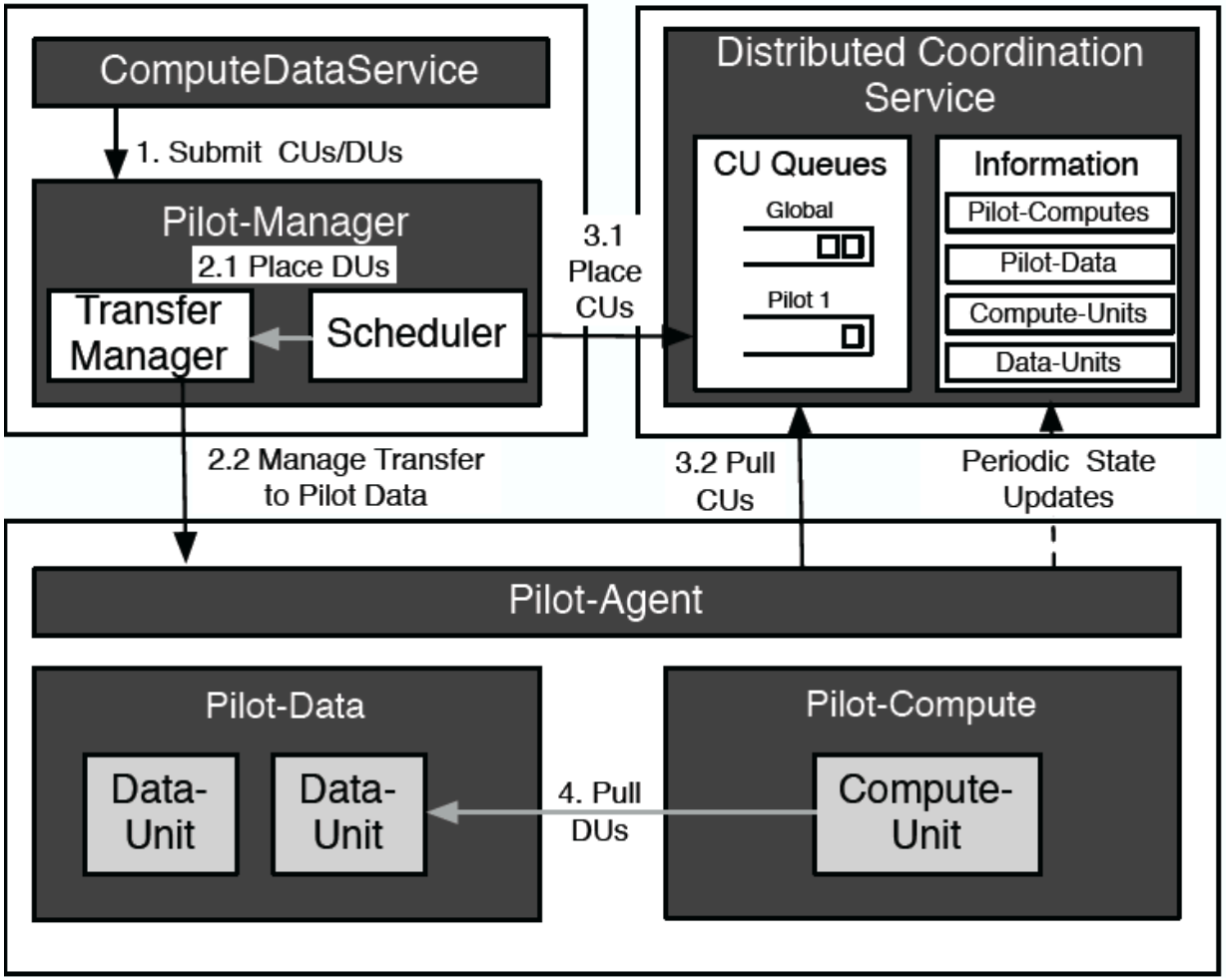
Understand terms:

http://saga-project.github.io/BigJob/sphinxdoc/usage/appwriting.html

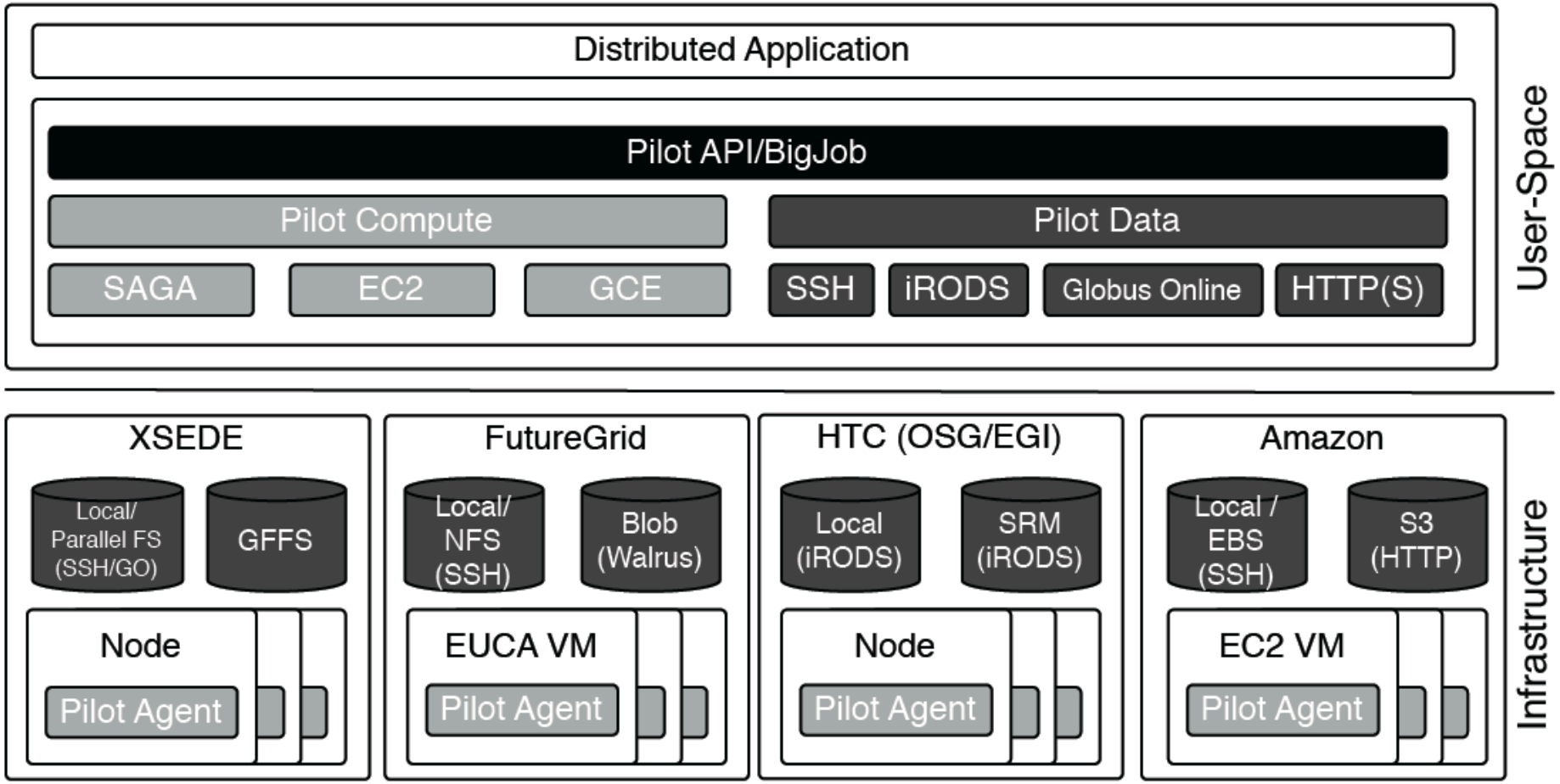
BigJob: Implementation of the P* Model



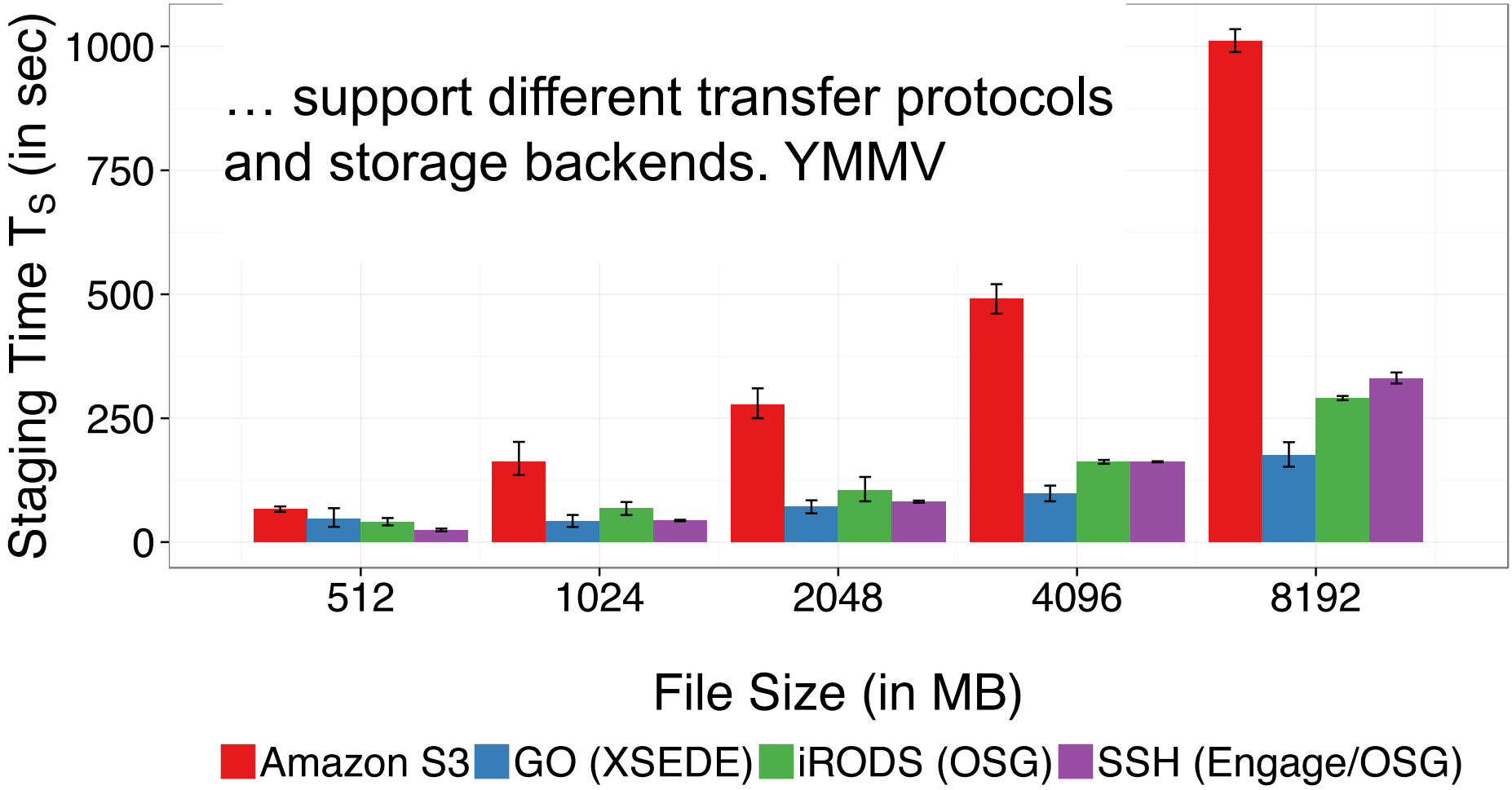
BigJob Workload Management



BigJob: Resource Interoperability



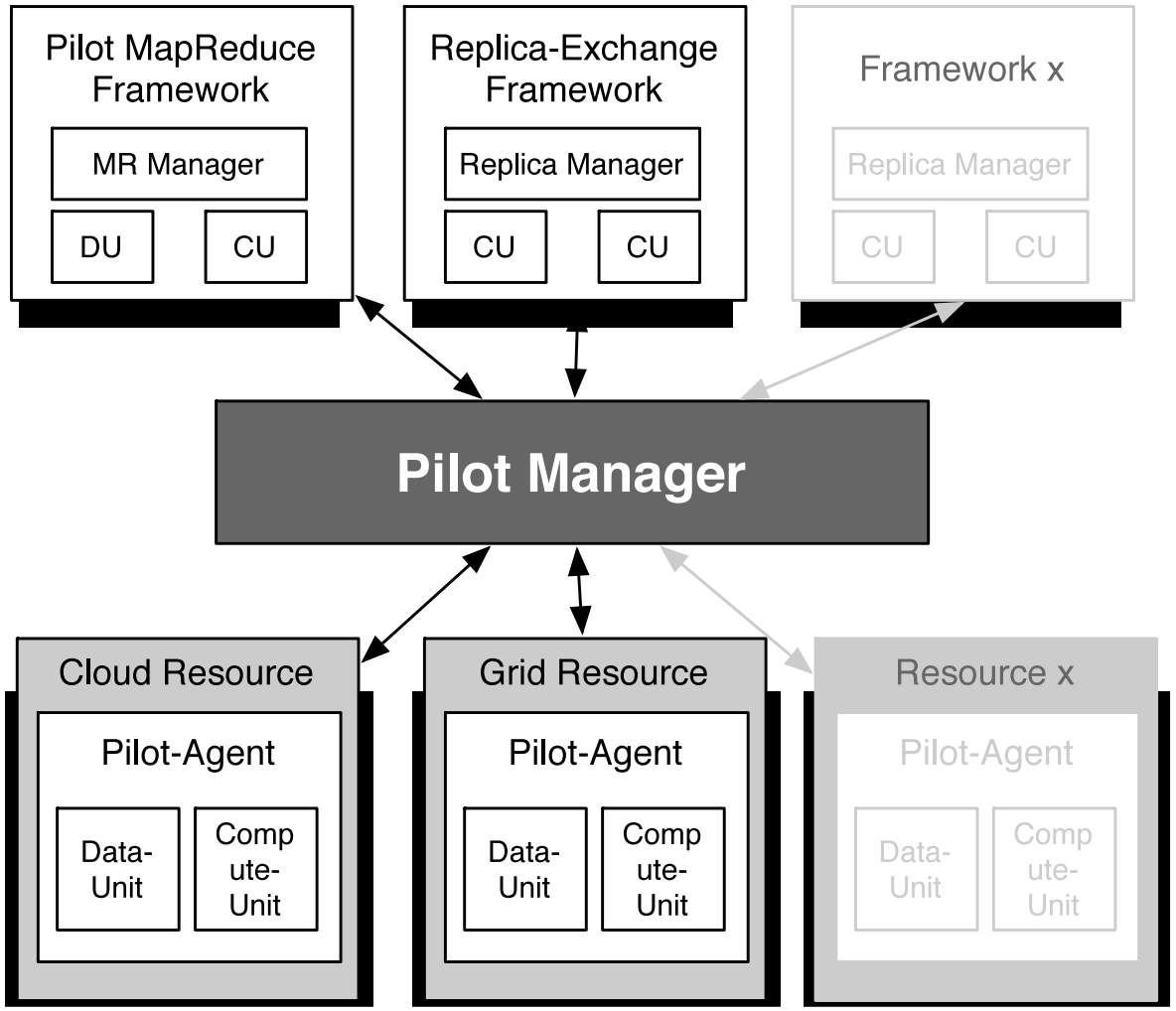
Pilot-Data Performance



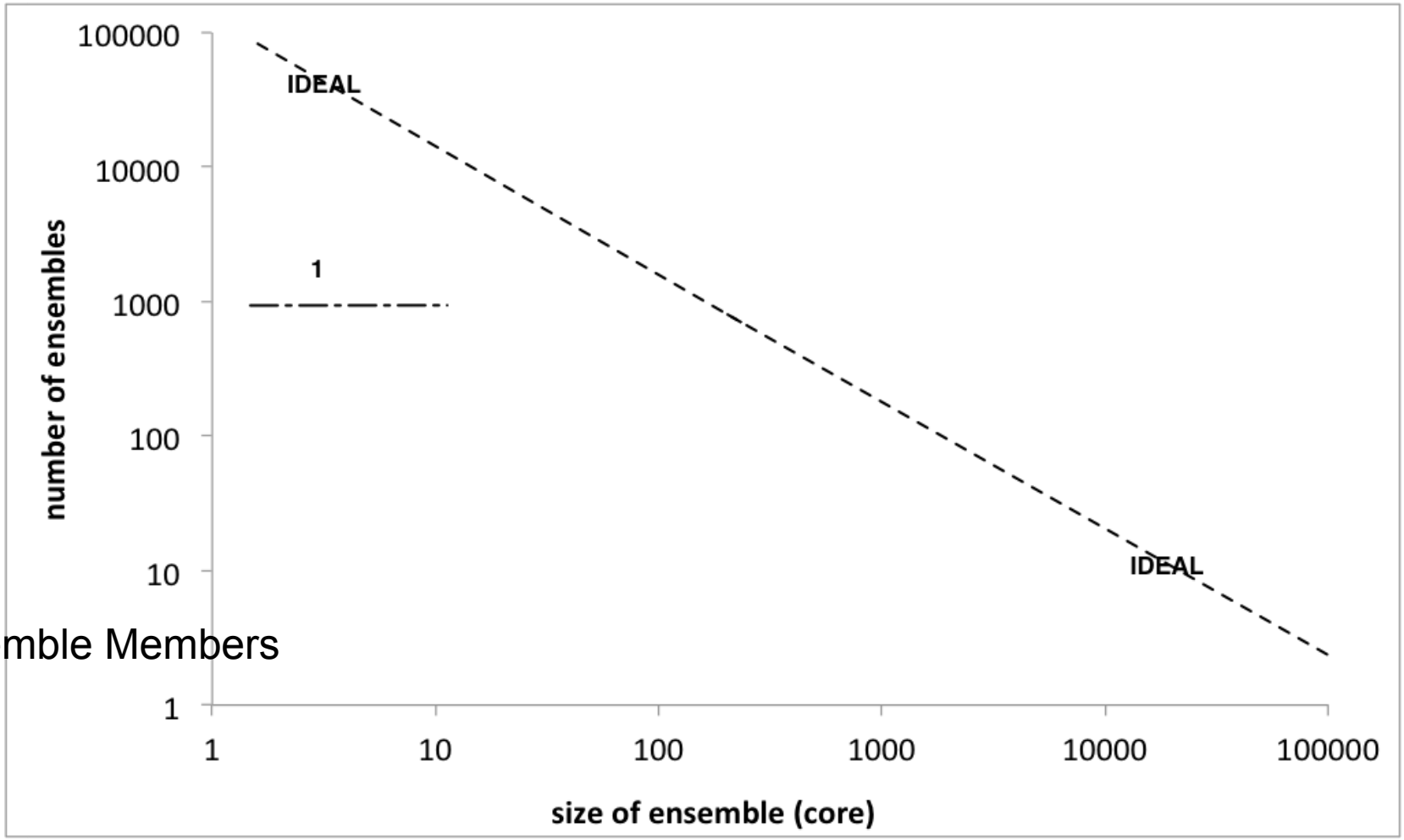
“Many Simulations”: Background and Scenarios

- A Single “Application” is broken into many smaller tasks
  - Naturally decomposed or “by design” (algorithmic and infrastructural)
  - Exploiting parallelism: coarse-grained
- Different types of coupling between these tasks
  - Uncoupled Tasks, Loosely-Coupled Tasks, Sequential Tasks..
- Considerations:
  - “Coupling” general concept for data sharing, synchronization, other dependency
  - Varying rate of coupling between tasks/simulations
  - Regular versus Irregular synchronization:
    - Temporal time
  - Ad hoc pair wise exchange
    - No a priori determined exchange partners
  - Varying task duration: hours to days to weeks

Pilot-Abstraction: Supporting Heterogeneous Application Workloads



“Slice and Dice” Resources



# of Ensemble Members

Number of cores per Ensemble Member

Scaling Along Many Dimensions

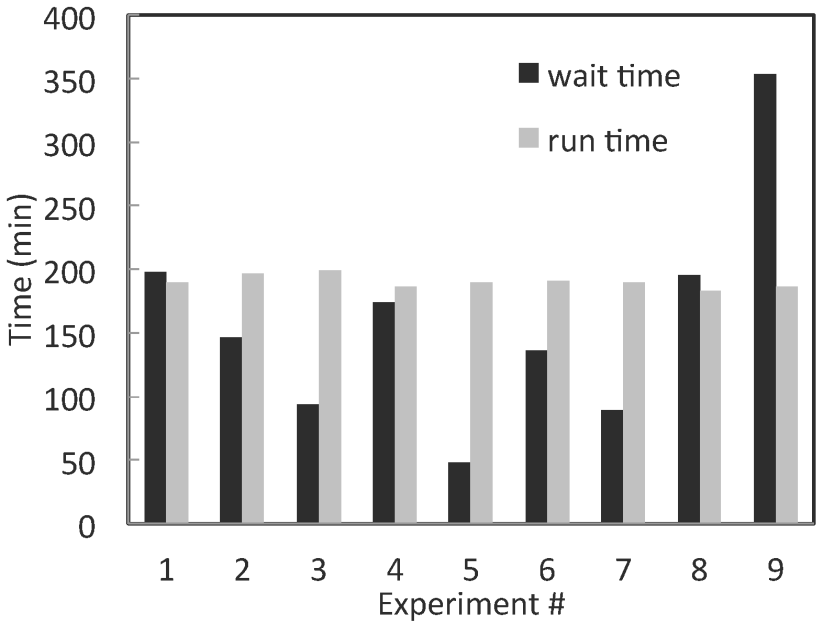
- Scaling Dimension #1: Size of each task
  - *Scale-up*
  - 1 core per task, 128 cores per task..
- Scaling Dimension #2: Total Number of Tasks
  - *Scale-Out*
  - Enhanced Sampling in molecular simulations O(1000), statistical errors O(10^6),…
- Scaling Dimension #3: Number of Resources Used
  - *Scale-Across*
  - Execute on Clouds, Clusters, Supercomputers
- Scaling Dimension #4: Total Number of Tasks/per unit time
  - Real time processing

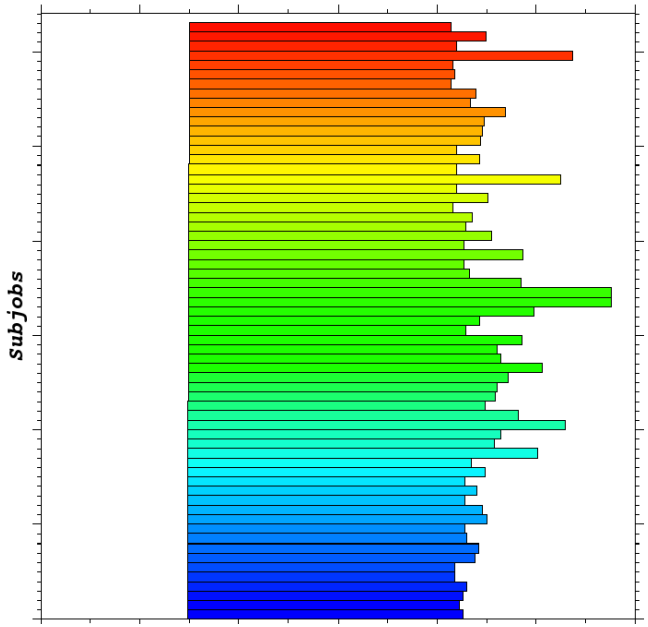
"Coarse-Grained" BigJob Performance

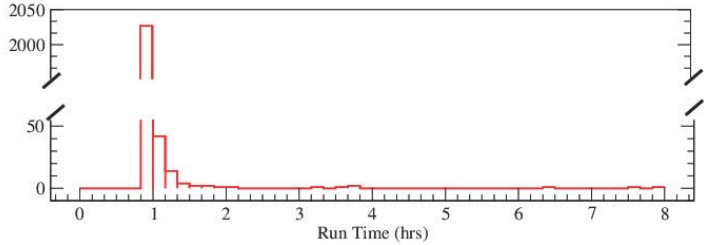
- Number of tasks that BJ can dispatch per second:
  - Distributed: O(10)
  - Locally: > O(10)
- Number of Pilots (Pilot-Agents) that can be marshaled
  - Locally/Distributed: O(100)
- Number of tasks concurrently managed:
  - Number of Pilot-Agents x Per each agent = O(100) x O(1000)
- Typical number of sub-jobs per Pilot-Agent:
  - Locally/distributed: O(1000)

- Obviously the above depend upon data per task:
  Range of data: O(1)--O(10^9) Bytes
- Duration of each task: O(1) second to O(10^5) seconds

Kraken: 126 ensembles, each of 192 cores = 24192 cores







References

- ``**P*: A Model of Pilot-Abstractions"**, 8th IEEE International Conference on e-Science 2012 (DOI: 10.1109/eScience.2012.6404423)

- *"***Distributed Computing Practice for Large-Scale Science & Engineering Applications"** *Shantenu Jha, Daniel S. Katz, Jon Weissman* et al, Computing and Concurrency: Practice and Experience, 2012 (DOI: 10.1002/cpe.2897)

- **"Pilot-Data: An Abstraction for Distributed Data"**, arXiv:1301.6228

References

- SAGA-Python:
  - http://saga-project.github.io/saga-python/
- BigJob: An implementation of P*
  - http://github.com/saga-project/BigJob/wiki
- RADICAL:
  - http://radical.rutgers.edu/
- Publications:
  - http://radical.rutgers/edu/publications

Acknowledgements

Graduate Students:

- Ashley Zebrowski
- Melissa Romanus
- Mark Santcroos
- Anton Trekalis

Undergraduate Students:

- Vishal Shah

Research Scientist/Programmer:

- Andre Luckow
- Andre Merzky
- Matteo Turilli
- Ole Weidner