RUTGERS

THE STATE UNIVERSITY OF NEW JERSEY

A Fresh Perspective on Pilot-Abstractions

Shantenu Jha

http://radical.rutgers.edu

Workshop on HPC and Super-computing for Future Science Applications

Brookhaven National Lab

05 June 2013

Outline

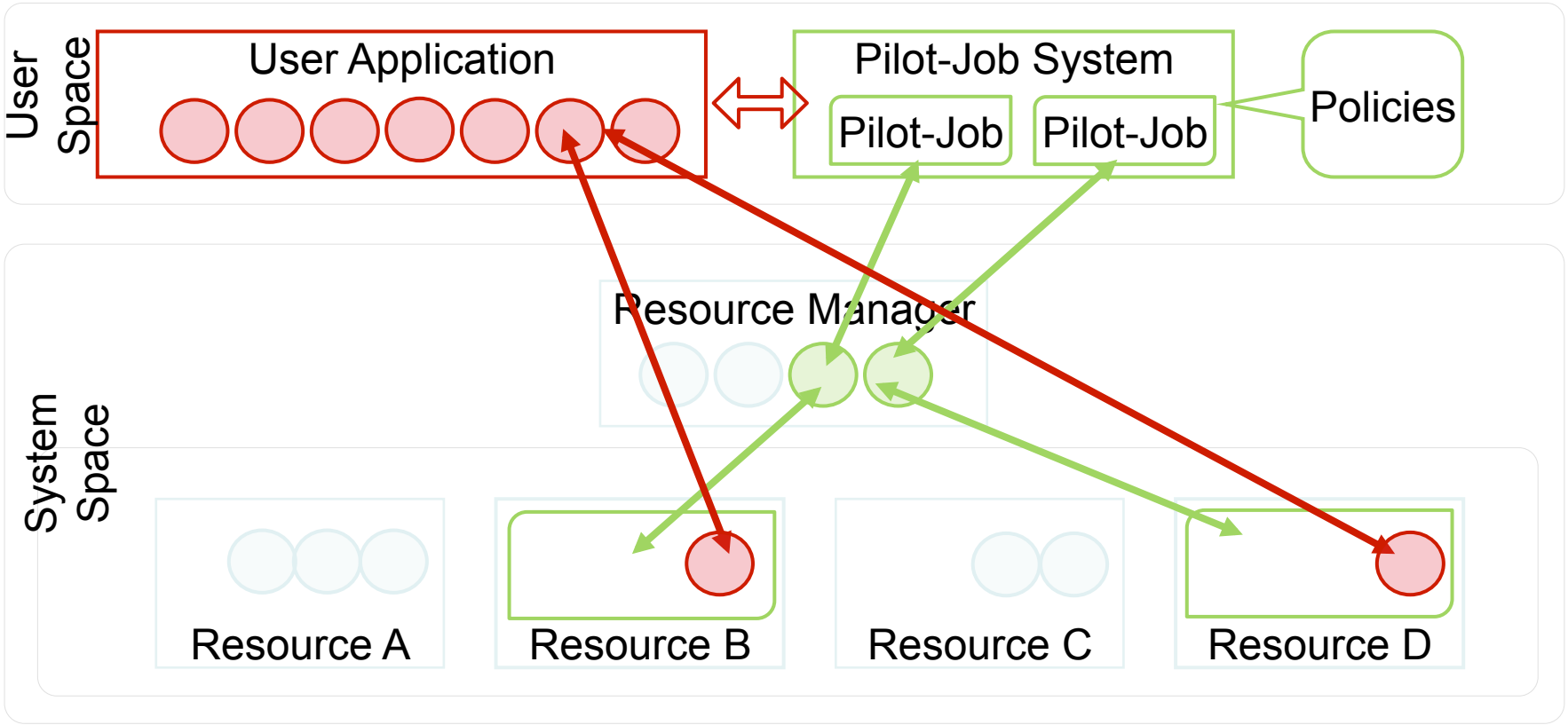
- Introduction to Pilot-Jobs
  - A View from the HPC end of the spectrum
    - "If the forward problem is hard, the reverse problem is not easy"!!

- Abstractions, Model and Implementation
  - Essence of Pilot Abstraction (Paradigm)
  - P*: A Model for Pilot
    - One possible model for Pilot Abstractions
  - BigJob: An implementation of P* Model
    - Example: Extensible and interoperability
    - Message: "Flexible" usage whilst meeting performance needs

- Quo Vadis? Consilience?

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

Pilot-Jobs (PJ): Five Myths

- PJs do not need well defined architecture, model and semantics, or PJs are such a simple concept, it doesn't need more “attention”
  - Not to confuse “simple to use” with simple to design”
- PJs are only about meta-scheduling (reducing queuing delays) on HTC, Or PJs unfairly game HPC queuing
  - There are interesting usage modes beyond “cycle stealing”
- PJ have to be tied to specific DCI; DCI are tied to specific PJ
  - Extensibility and interoperability have been difficult to establish
- PJs are passive (system) tools, as opposed to user-space, active and extensible components of a CI
  - PJs can be user-controlled “programmable elements
- PJ do not help with next-generation “data-intensive” applications
  - PJ for NGS O(10-100) GB per task on existing DCI

Landscape of Pilot-Jobs

- There are many Pilot-Job offerings, often semantically distinct
  - PanDA, DIANE, DIRAC, Condor Glide-In, SWIFT, ToPoS Falkon, BigJob…
    - *Why do you think there has been a proliferation of PJs?*

- Conceptual & practical barriers to extensibility (& interoperability)
  - The landscape of PJ reflects, in addition to PJ specifics, the broader eco-system of distributed middleware & infrastructure
  - Software Engineering issues, interfaces, standardization

- Difference in the execution models of the PJs
  - We know "what" pilot-jobs do, but the "how" remains less clear
    - How to map tasks to pilot-jobs? How to choose/map optimal resource?
    - How to "slice and dice" resources?

- Data remains a dependent variable, not a primary variable
  - Introduce the concept of Pilot-data

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

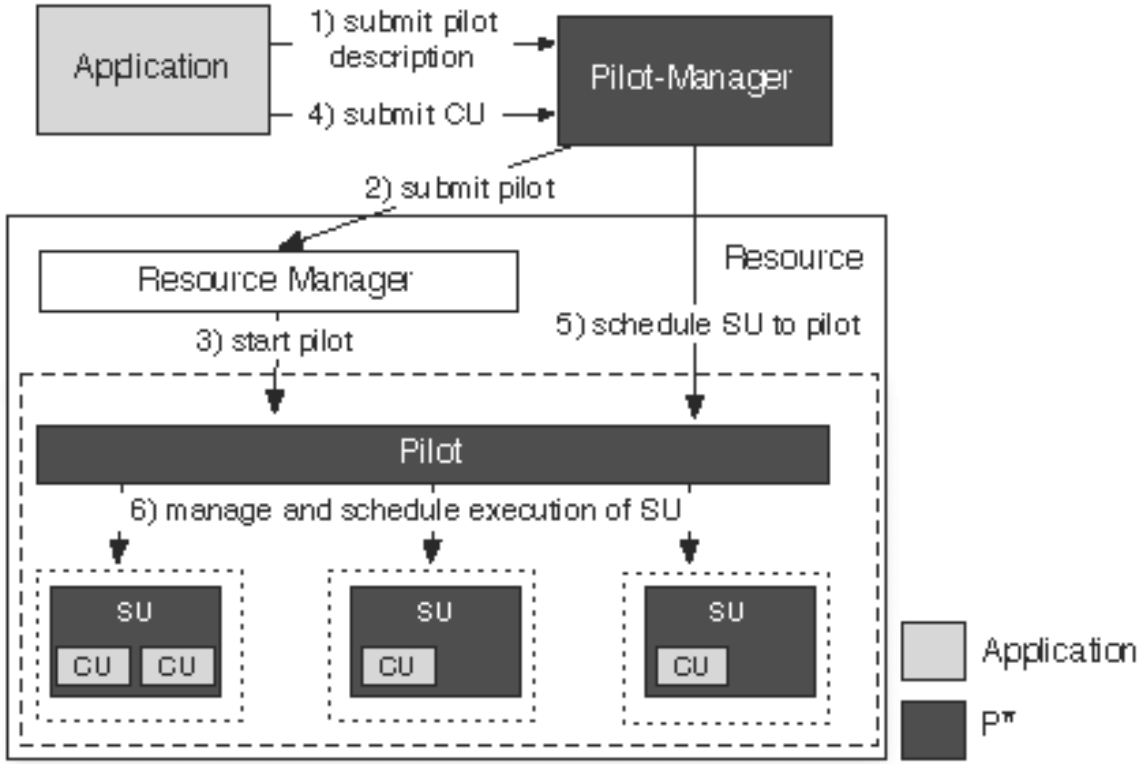
P*: A Conceptual Model for Pilot Abstractions

- A Minimal but complete model
  - **Minimal**: Towards a common understanding of pilot-jobs
    - Provides vocabulary and model for analysis and insight across different PJs
    - Does not try to provide closed form answers to all issues
  - **Complete:** Can be used to design a priori an effective Pilot-Job

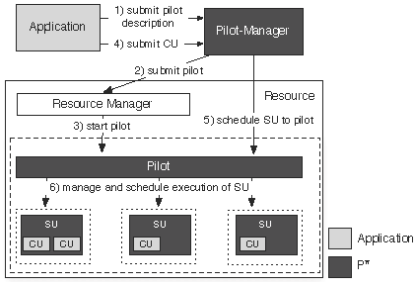
- Establish basic terminology, functionality and inter-relationship
  - What is the pilot? What is the agent? Push or pull?
- Provide a framework for comparison across Pilot implementations
  - Many existing pilot implementations can be understood using P*
  - Not all pilot implementations must adhere strictly to P*
- Unified view of Pilot-Abstractions
  - Provides symmetrical treatment for compute and data (and eventually network)

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



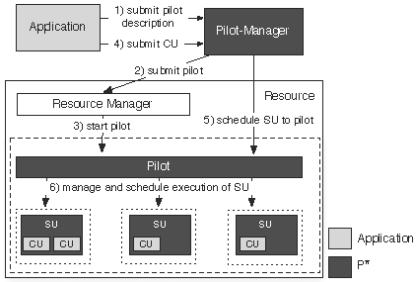
Pilot-Compute

- 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



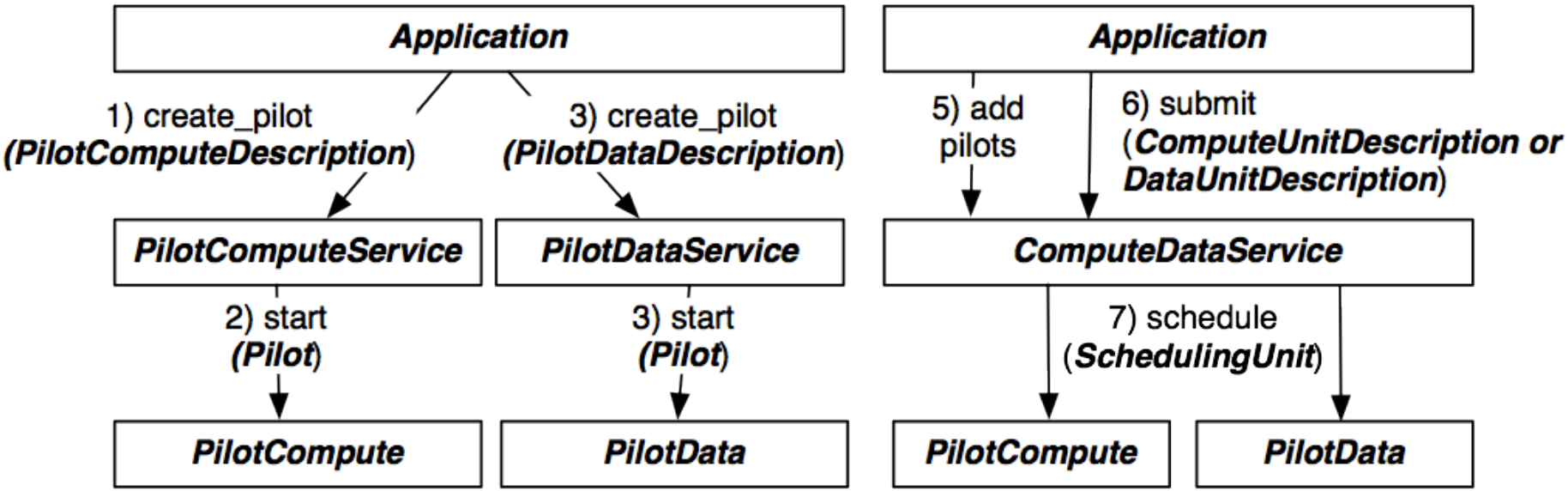
Compute Unit (CU)

- 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

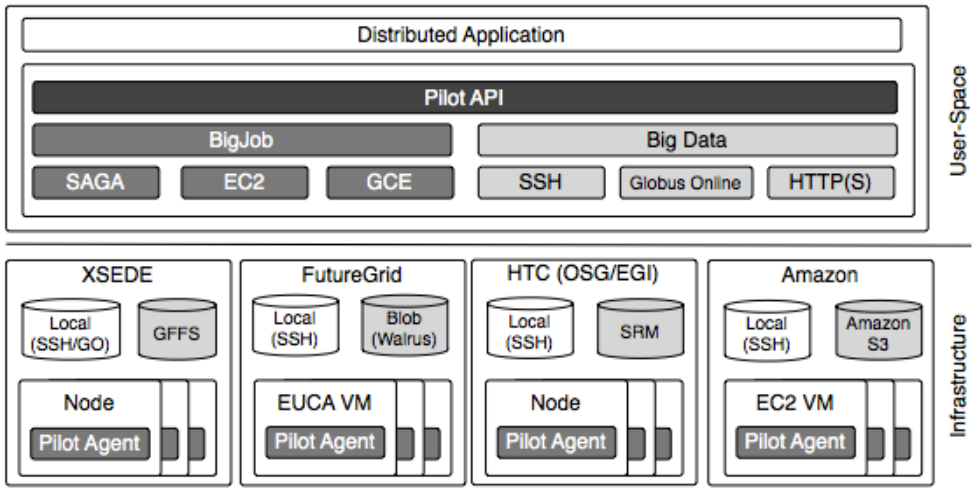


Managing Pilots

Managing Application Workload

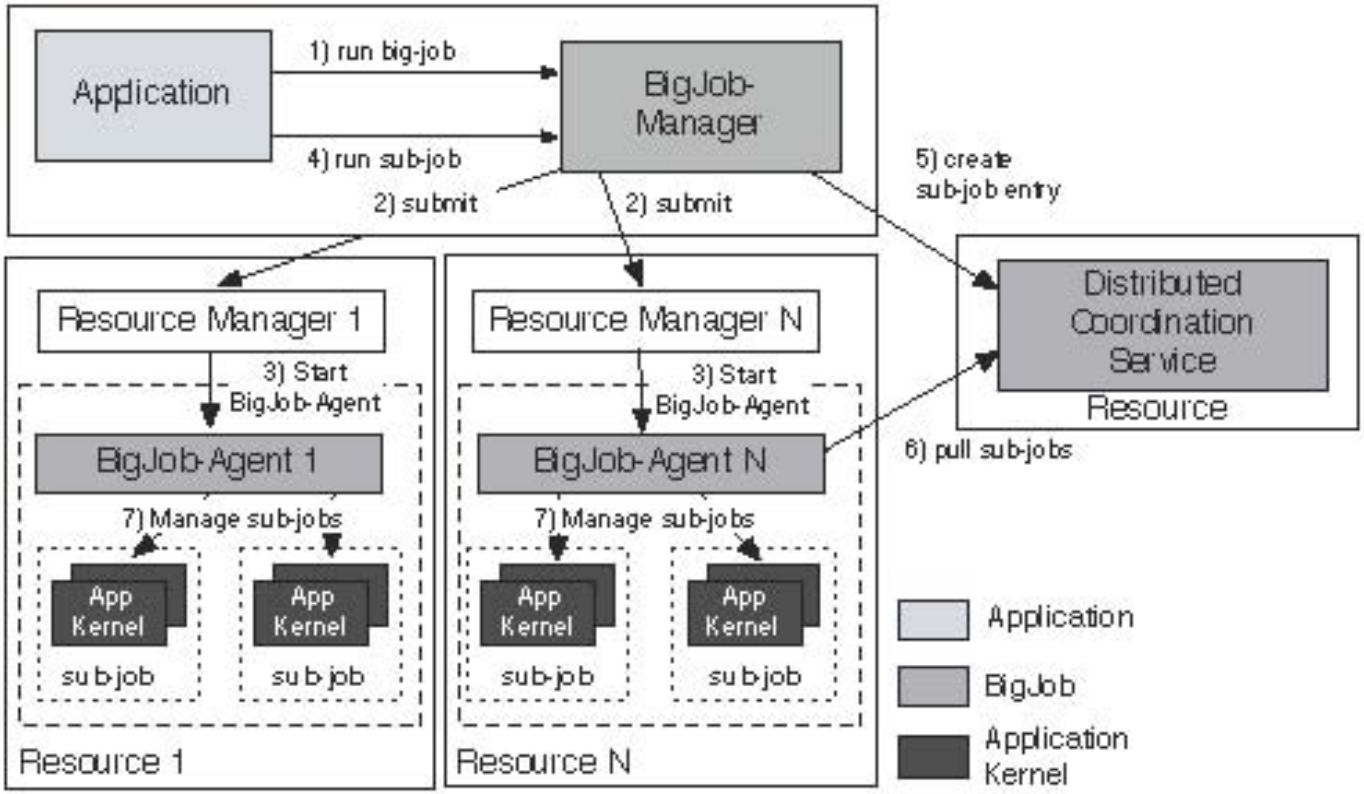
P*: Mapping and Interoperability

P* Element	BigJob	DIANE	Condor-G/ Glide-in
Pilot-Manager	BigJob Manager	RunMaster	condor_master condor_collector condor_negotiator condor_schedd
Pilot	BigJob Agent	Worker Agent	condor_master condor_startd
Compute Unit (CU)	Task	Task	Job
Scheduling Unit (SU)	Sub-Job	Task	Job

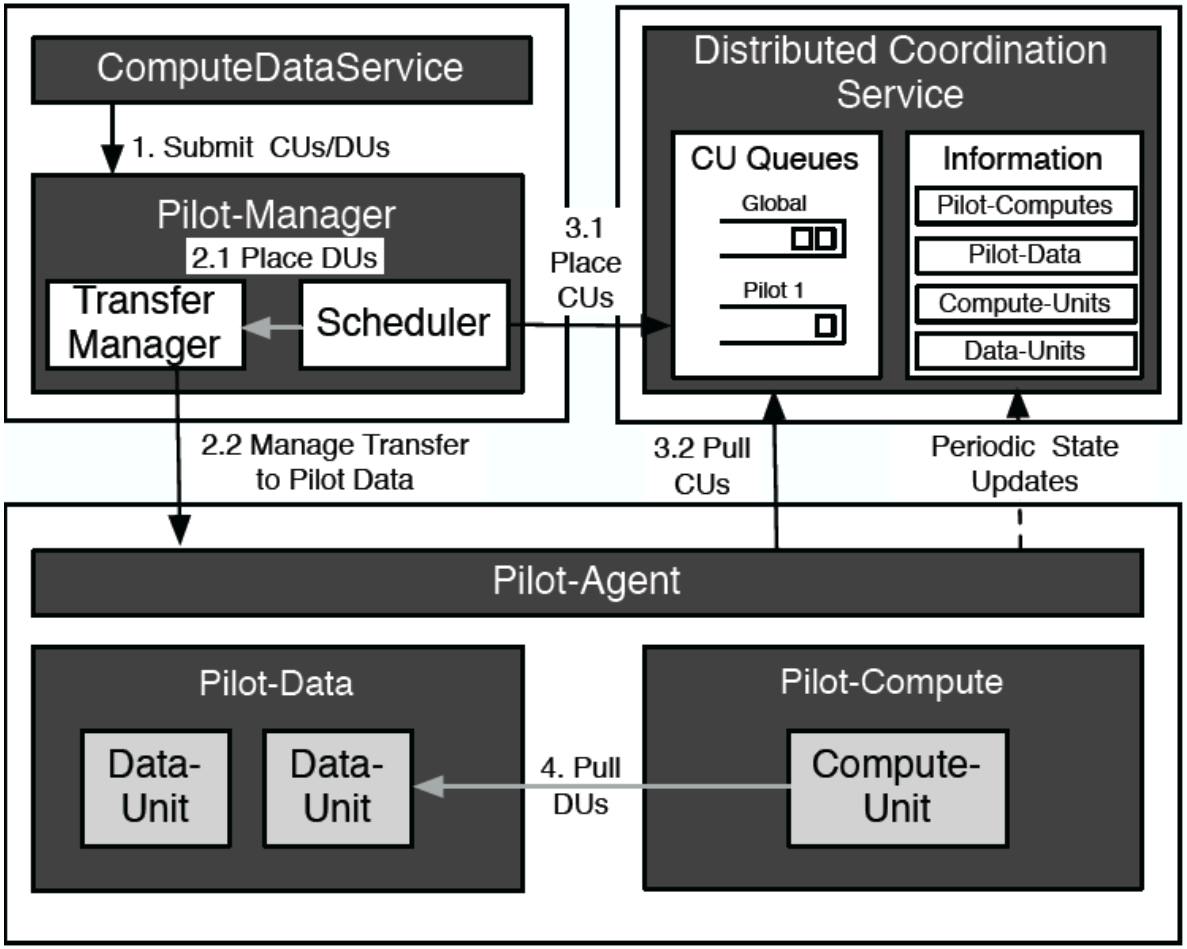


BigJob: A Reference Implementation of the P* Model

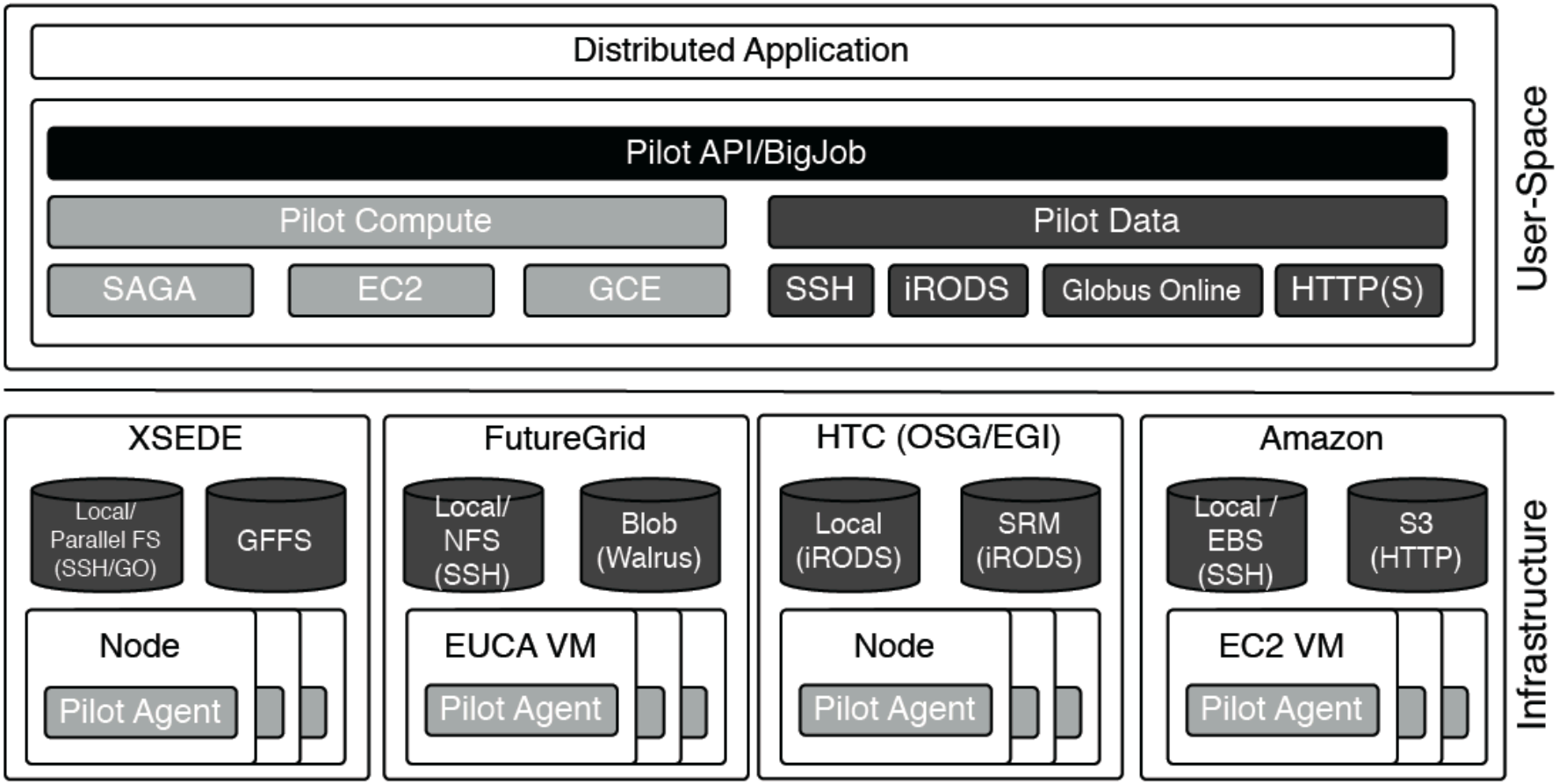
BigJob: Implementation of the P* Model



BigJob Workload Management



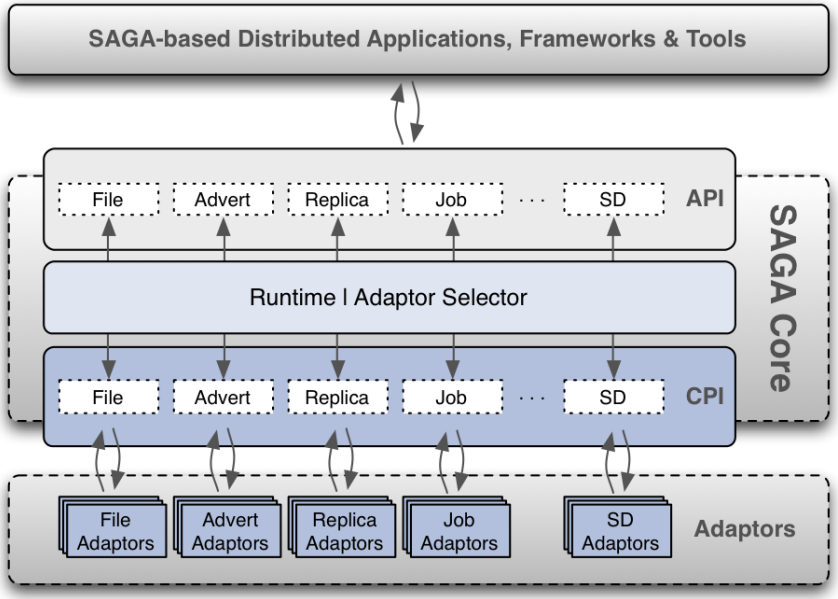
BigJob: Resource Interoperability

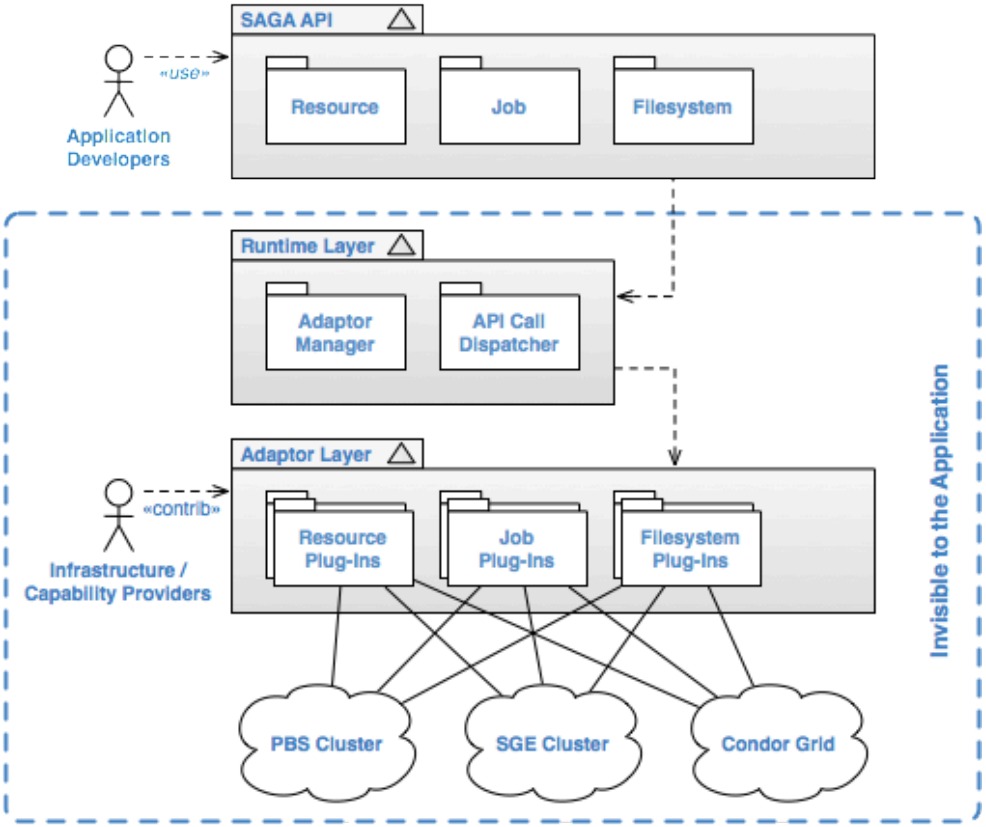


SAGA: Interoperability Layer for BigJob

http://saga-project.org

SAGA: Standard for Distributed Applications



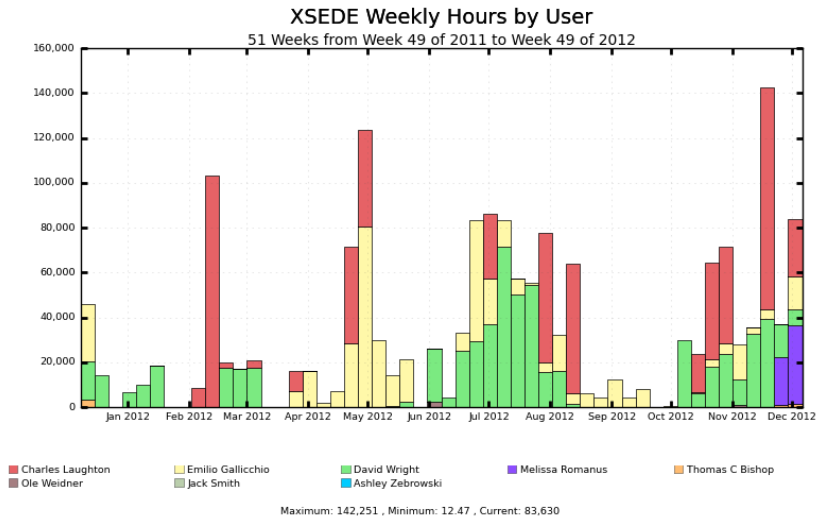


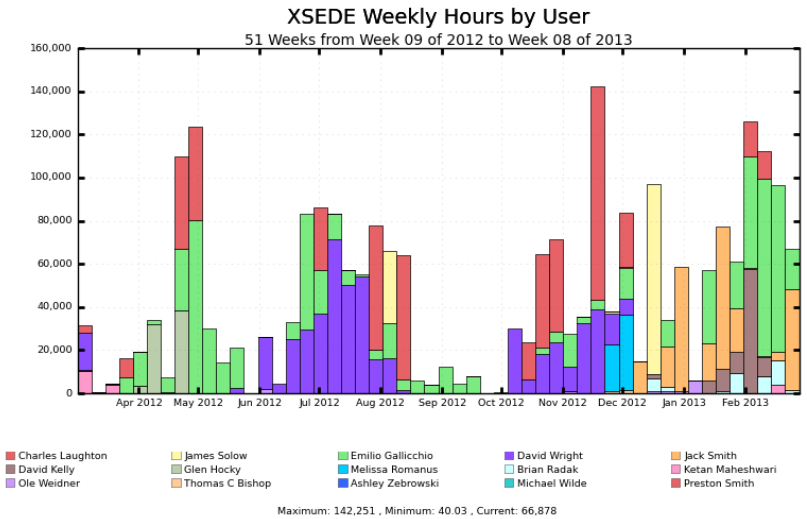
SAGA: Interoperability layer upon which other tools and applications are built

- HOW SAGA is Used?
  - Uniform Access-layer to DCI
    - EGI, XSEDE, DATAONE, UK NGS and NAREGI/RENEKI and Clouds
  - Application "Scripting Layer" to DCI
  - Build tools, middleware services and capabilities that use DCI
    - e.g. Gateways, *Pilot-Jobs*
- WHAT is SAGA Used for?
  - Support production-grade science and engineering
  - Research tool to design, implement reason about distributed programming models, systems and applications

BigJob: An Extensible, Interoperable Pilot-Job for HPC and HTC Workloads

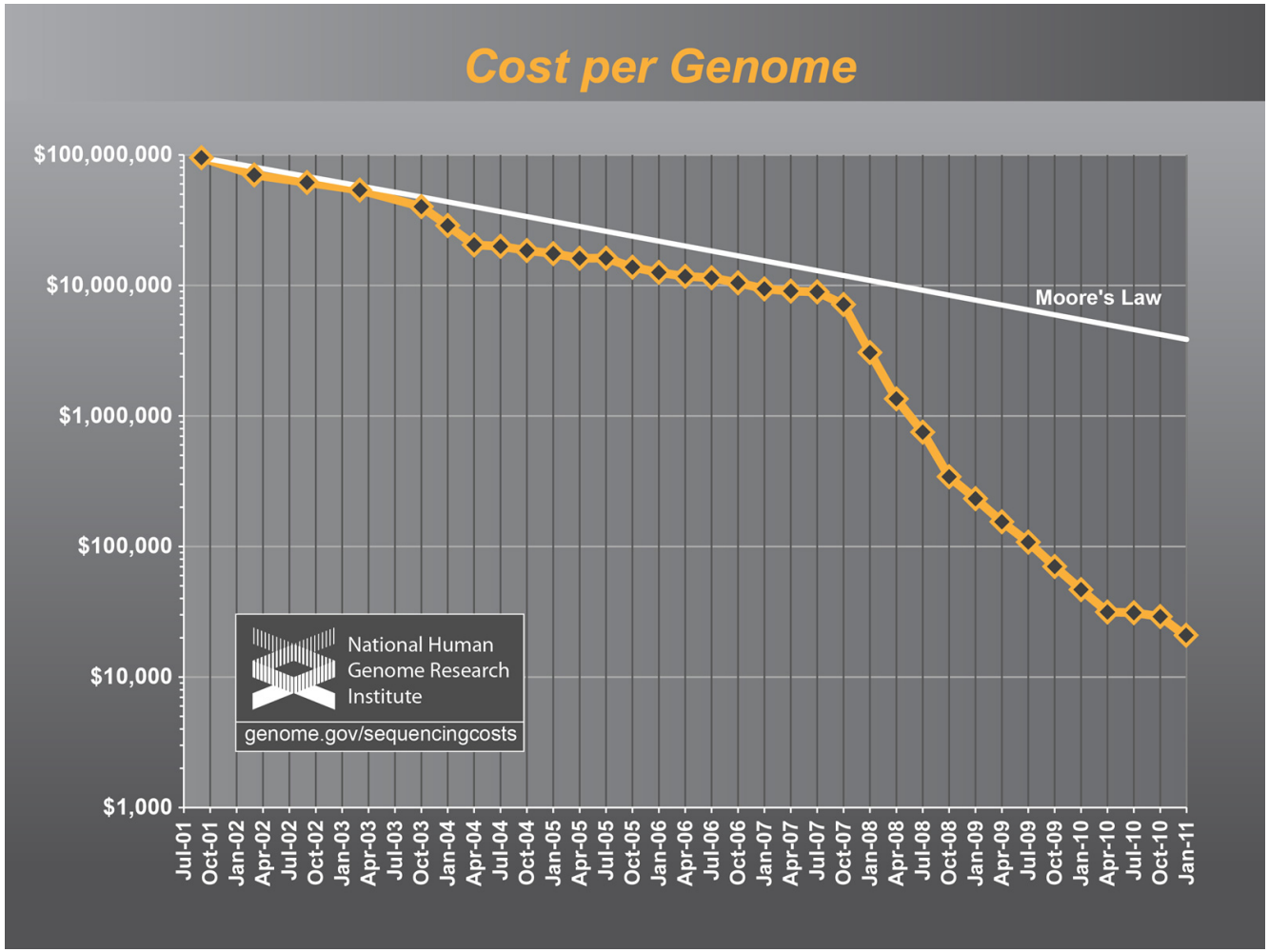
BigJob: (Partial) Usage on XSEDE Machines



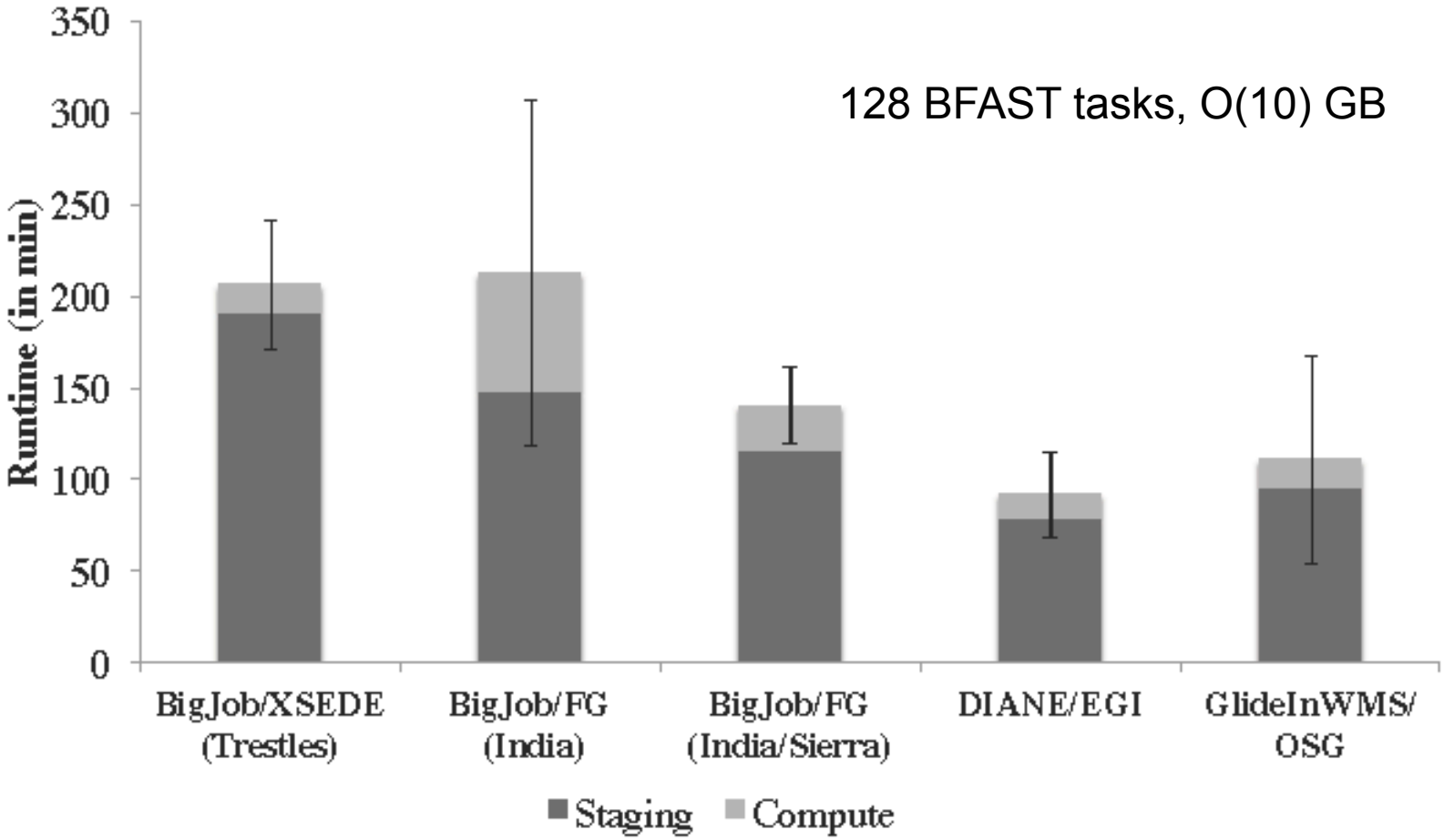


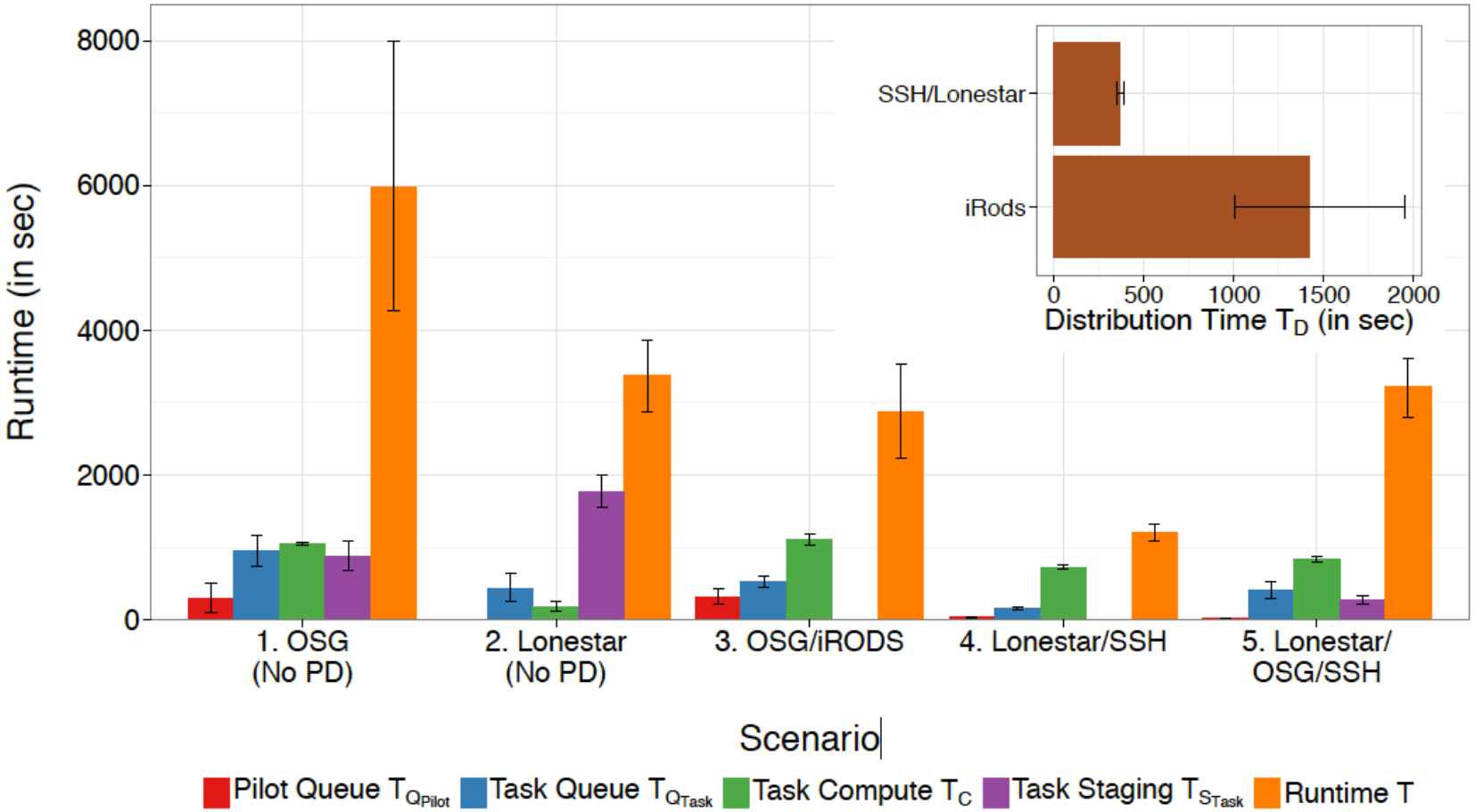
> 10M SUs/year (and increasing) on XSEDE machines

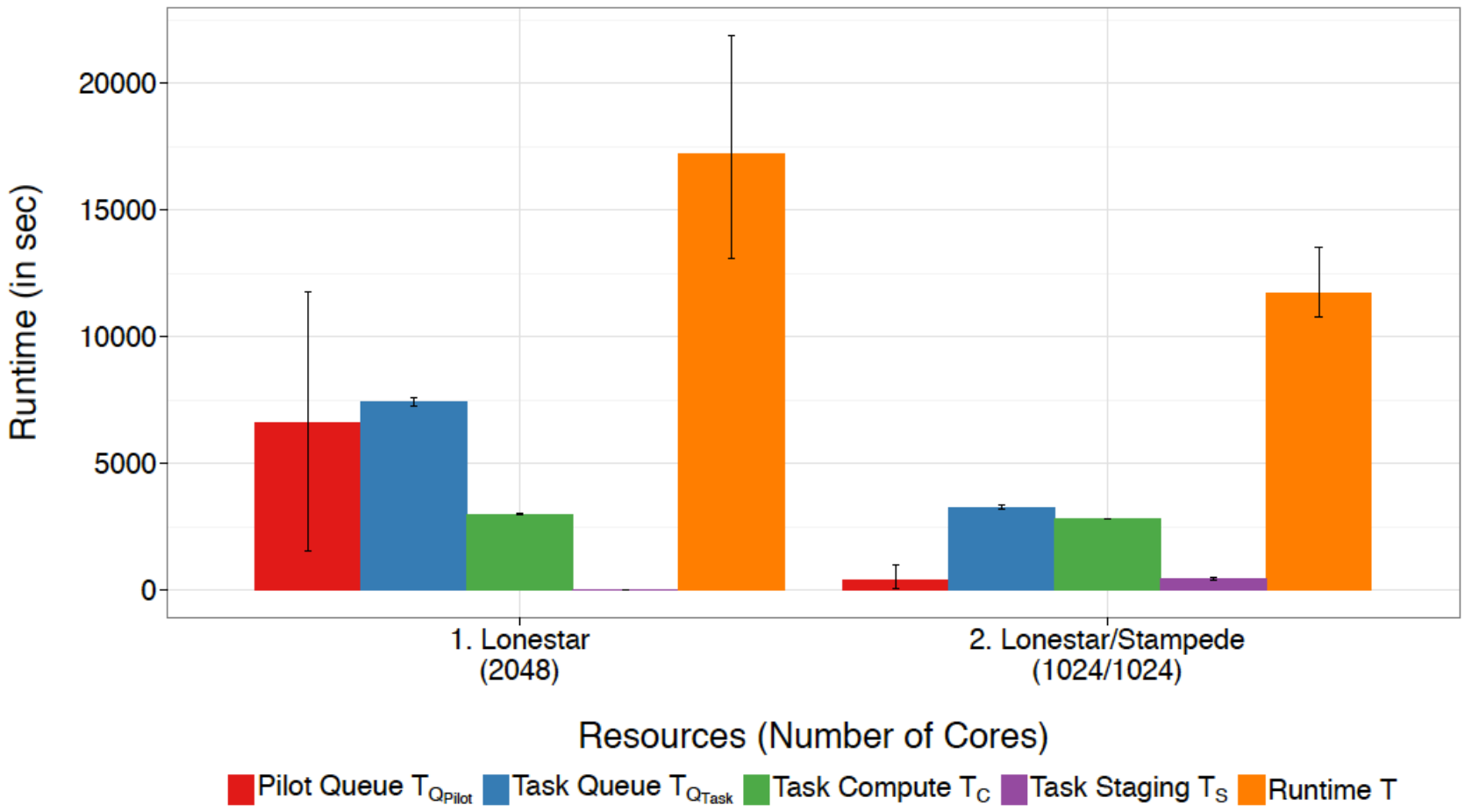
The Challenge of Integrating Compute and Data at Scale



Accessing Multiple DCI & Pilot-Jobs via Pilot-API




Runtime (in sec)
8000
6000
4000
2000
0
1. OSG (No PD)
2. Lonestar (No PD)
3. OSG/iRODS
4. Lonestar/SSH
5. Lonestar/ OSG/SSH
Scenario
SSH/Lonestar
iRods
0
500
1000
1500
2000
Distribution Time T_D (in sec)
Pilot Queue $T_{Q_{Pilot}}$
Task Queue $T_{Q_{Task}}$
Task Compute T_C
Task Staging $T_{S_{Task}}$
Runtime T


Runtime (in sec)
20000
15000
10000
5000
0
1. Lonestar
(2048)
2. Lonestar/Stampede
(1024/1024)
Resources (Number of Cores)
Pilot Queue $T_{Q_{Pilot}}$
Task Queue $T_{Q_{Task}}$
Task Compute T_C
Task Staging T_S
Runtime T

"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)

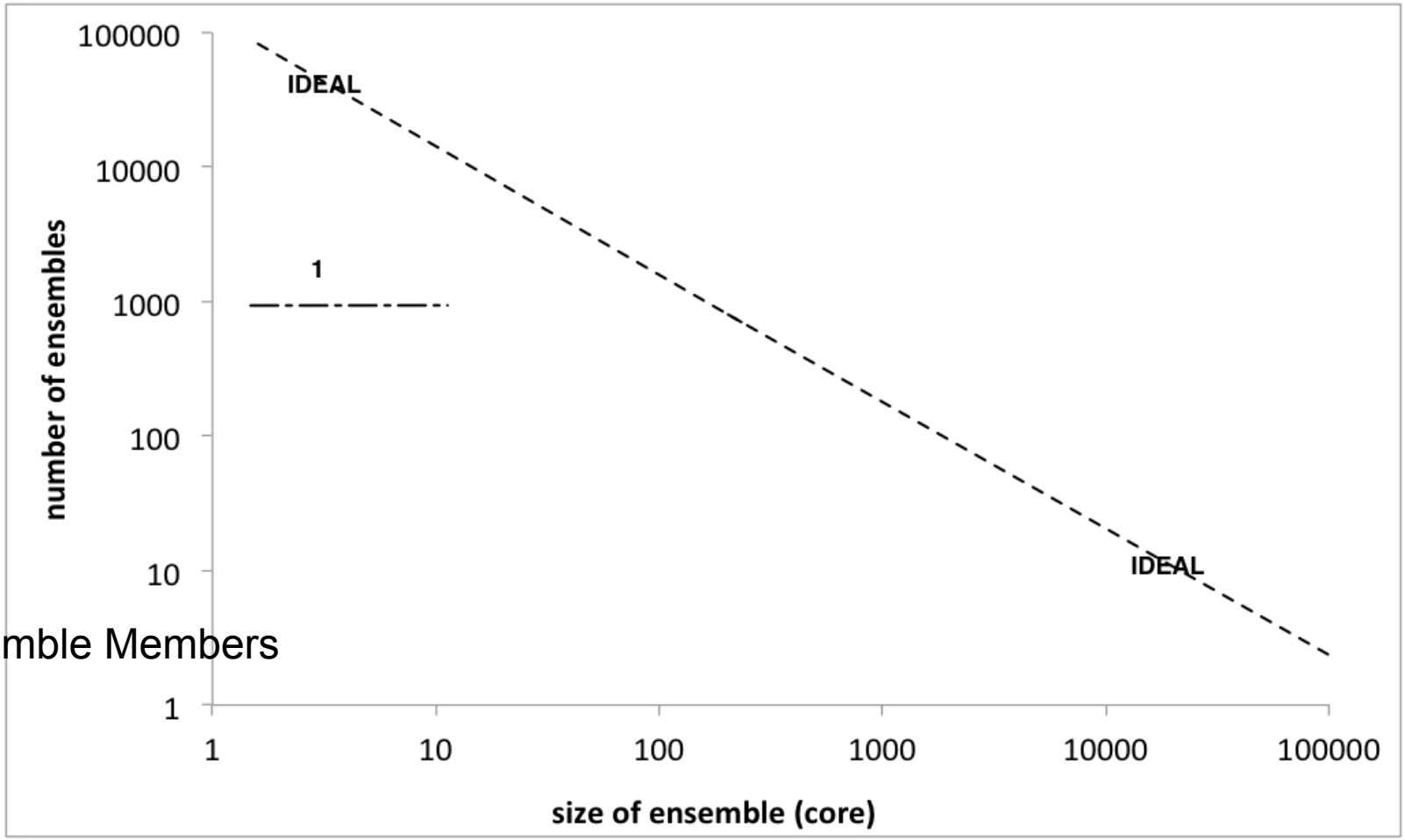
- Obviously the above depend upon data per task:
  Range of data: O(1)--O(10^9) Bytes
- Duration of each task: O(1) second to O(10^5) seconds

BigJob Exemplar: Varying the “coupling” between many MD Simulations on Many Supercomputers

Uncoupled and Loosely Coupled Parallel Molecular Simulations

- Enhanced Sampling of Uncoupled Ensembles
  - Large Ensembles of HPC simulations, Multi-stage/chaining of HPC simulations
- Loosely Coupled: Understand protein-ligand recognition via (multidimensional) replica-exchange
- Requirements:
  - Launch and monitor/manage (order 10^2-10^4) ensemble members
    - Where each ensemble member could be 128-1024 cores
  - Varying degrees of coupling between ensemble members
  - Varying job duration: hours to days to weeks
  - Ad hoc pair wise (a)synchronous data exchange
    - No global synchronization, No a priori determined exchange partners

HP-HTC or HT-HPC?

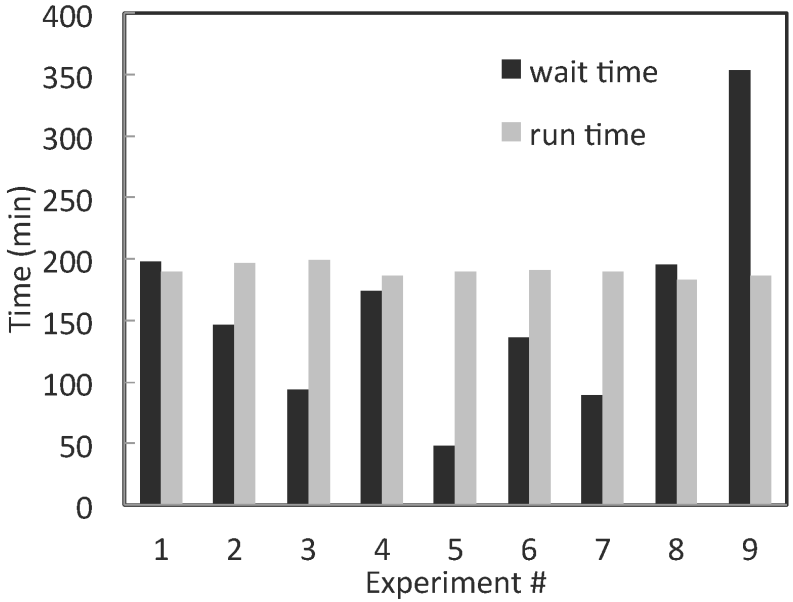


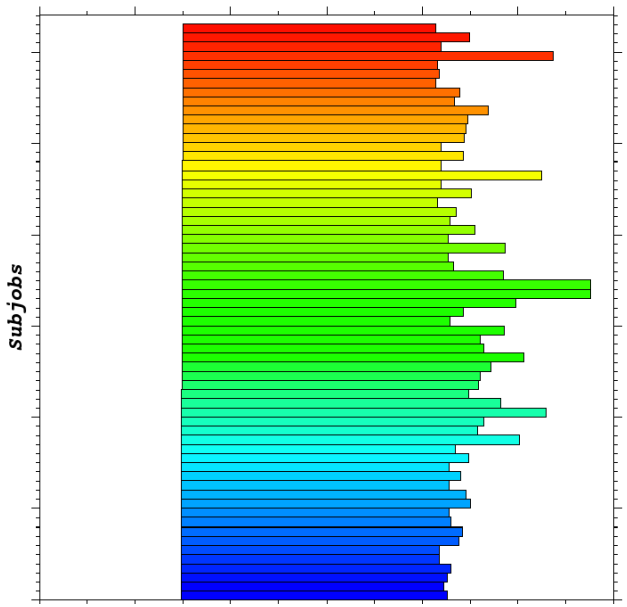
# of Ensemble Members

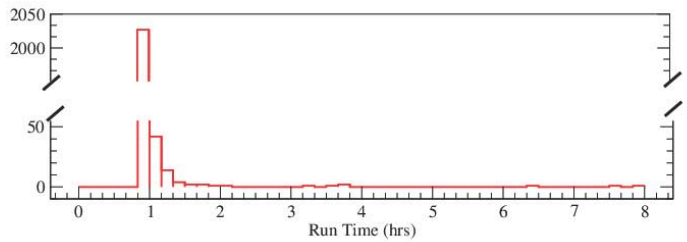
Number of cores per Ensemble Member

HT-HPC on Kraken

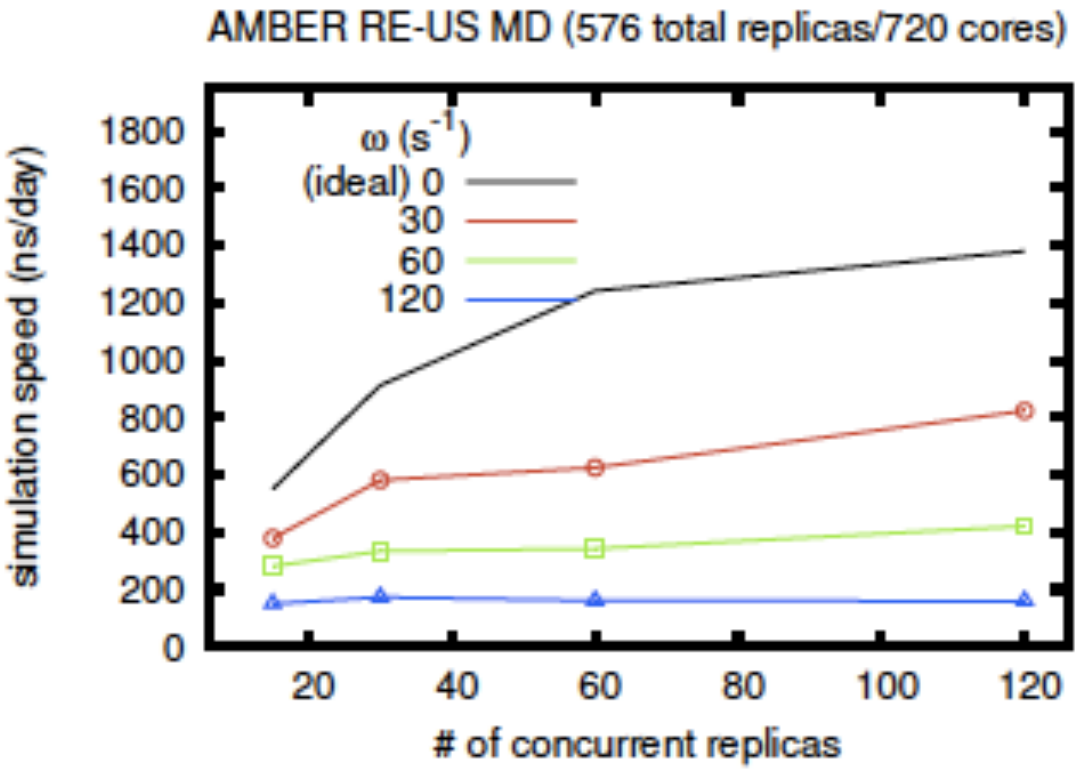
126 ensembles, each of 192 cores = 24192 cores





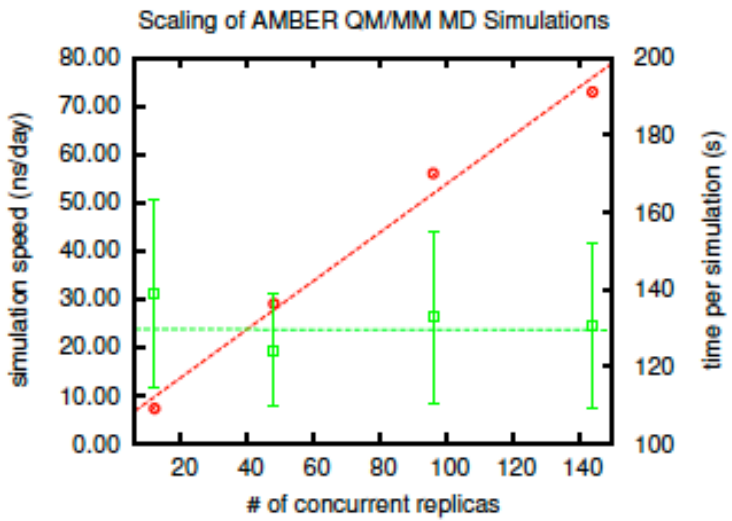


Replica-Exchange Performance in AMBER



Umbrella sampling of the backbone conformational space of alanine dipeptide. The exchange parameters are all permutations of harmonic biasing potentials on each torsion.

QM/MM Replica-Exchange



- QM/MM is not generally considered parallelizable
  - RE provides efficient sampling irrespective
- Repex FW: BigJob-based Repex
  - Framework to handle O(100)--- O(1000) concurrent simulations
  - Amongst the earliest QM/MM
  - Framework similar to classical QM/MM
- Performance increases linearly with core count and number of QM/MM simulations
  - No apparent coordination cost for additional simulations

RUTGERS

DoE AIMES: Abstractions and Integrated Middleware for Extreme Scale

http://aimes-project.org

DoE-ASCR (DE-FG02-12ER26115)
Collaboration between Rutgers (lead), Chicago and UM

Overview

- The Assumption:
  - Distributed computing is one pathway to extreme-scale and a necessity for collaboration
- The Problem
  - Current state of distributed applications and infrastructure is problematic
  - Two situations predominate:
    - Applications are inflexible: optimized and pinned to a specific platform
    - Applications are flexible: run anywhere but performance is unpredictable
- The Hypothesis
  - Flexibility and performance are compatible!
  - Extracting Simplicity whilst managing Complexity requires abstractions
    - Abstractions and Models are key to “reason about distribution”
- The Solution
  - We advance abstractions that will facilitate reasoning integrated across application and resource level
- AIMES: Integrated approaches to Resource Management

Conclusion

- "Pilot Abstraction Works!"
  - Different application types, execution models and requirements
  - Execution on traditional (HPC/HTC) and emerging platforms

- Towards "general purpose" pilots? Extensibility and Interoperability will need to be first order concerns
  - Pilot Abstraction → P* Model → BigJob
  - Commonality across Pilot Implementations ?
    - Seek convergence and coordination, not domination!
    - Best practices? Common APIs/Interfaces? Standardization?

- Extreme Scale Science requires abstractions and models
  - Using abstraction and models, even (pilot) prototypes can deliver flexible usage yet meet performance requirements
  - AIMES: Abstractions as basis for "Integrated" middleware

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

- **"A Fresh Perspective on Pilot-Jobs"**, Review/Survey Article on Pilot-Jobs, to be submitted (2013)

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