SAGA
Simple API for Grid Applications

Abstractions for Life-Science Applications on Clouds

Shantenu Jha, Andre Luckow

Based on work by Joohyun Kim, S Maddineni, P Mantha, Mark Santcroos, Ole Weidner

http://saga.cct.lsu.edu

Microsoft Cloud Futures Workshop, 2011

SAGA
Simple API for Grid Applications

Overview

- Lessons from a decade of Developing Distributed Applications [PAST]
  - Unique Role for Abstractions for Distributed "Dynamic" Applications
- Understand the present challenges for LS Applications on Cloud
  - Understanding *common* computational characteristics
  - Motivate and Introduce abstractions for dynamic execution
    - "Autonomic" and "Intelligent" Pilot-Job
- Compute Intensive: EnMD and RE Simulations
  - Azure Solution: Architecture, Performance and Scalability
    - Azure addresses several of the distributed application challenges
- Data Intensive: NGS Analytics using BFAST [FUTURE]
  - XD/FutureGrid Solution: Architecture, Performance and Scalability
  - Lessons and Experience from DARE-based Gateway on XD/FG
    - Towards NGS Analytics as a Service on Azure?

Cloud Past: Lessons from Past Decade of Developing Distributed Applications

- Space of Distributed Applications (DA) Is large (and rich), but the number of effective and extensible DA small
  - More than just submitting jobs here and there!
- Developing DA is a hard undertaking
  - Coordination across resources & Execution Environment
  - Large number programming systems, tools & "incomplete solutions"
- Think "distribution", not hide from it!
  - Embrace distribution, e.g., data-centric application drivers!
  - Heterogeneity & dynamic execution are fundamental
- Point to a unique role for Pattern-oriented and Abstractions-based Development of Distributed Applications
  - Application & System-level Abstractions for Development, Deployment & Execution
    - "Abstractions allows innovation at more interesting layers"

SAGA
Simple API for Grid Applications

Assertion #2: Developing DA is a hard undertaking

- Intrinsic reasons why developing DA is fundamentally hard:
  - Control & Coordination over Multiple & Distributed sites
    - Effective coordination in order for whole > sum of the parts
  - Complex design points; wide-range of models of DA
    - Many reasons for using DA, more than (just) peak performance
- Extrinsic:
  - Execution environments will be dynamic, heterogeneous and expose varying degrees-of-control
    - Fundamental different variation in role of Execution Environment- distinguishing feature of DA from "regular environment" HPC
  - Application types strongly coupled to the infrastructure capabilities, abstractions/tools, & policy:
    - Often development tools assume "specific" deployment and execution environments, or don't where needed!
    - Policies, infrastructure & tools, e.g production DCI has been missing for DDDAS

Assertion #3: Think Distribution

- "History of computing like pendulum, swings from centralized to distributed"
  - Indications this time there is a fundamental paradigm shift due to DATA
  - Too much to move around; learn how to do analytics/compute *in situ*
- Decoupling and delocalization of the producers-consumers of computation
  - Localized special services; people and collaborations are distributed
- (Ironically) Most applications have been developed to hide from heterogeneity and dynamism; not embrace them
  - Programming models that provide dynamic execution (opposed to static), address heterogeneity
    - *"The reason why we are so well prepared to handle the multi-core era, is because we took the trouble to understand and learn parallel programming"* – Ken Kennedy

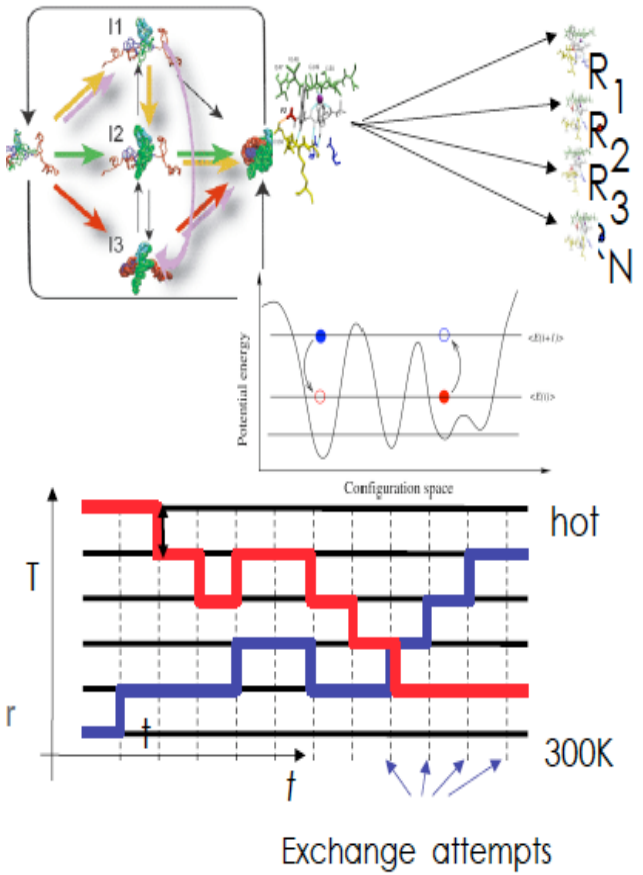
Clouds Present: Novel or more of the same?

- Clouds address several "barriers" of decade past
  - Better control over software environment via virtualization
  - Illusion of unlimited and immediate available resource can lead to better capacity planning and scheduling
- Clouds do not remove many/all of the challenges inherent in DA
  - Clouds are about provisioning, grids are about federation
  - Fundamental challenges in logical and physical distribution remain
    - Makes some thing worse as impose a model of strong localization
- If Clouds part of a larger, richer distributed CI
  - Certain tasks better suited for Grids, others on Clouds
- Clouds represent a natural and positive evolution but will need a careful interplay of application and system-level abstractions

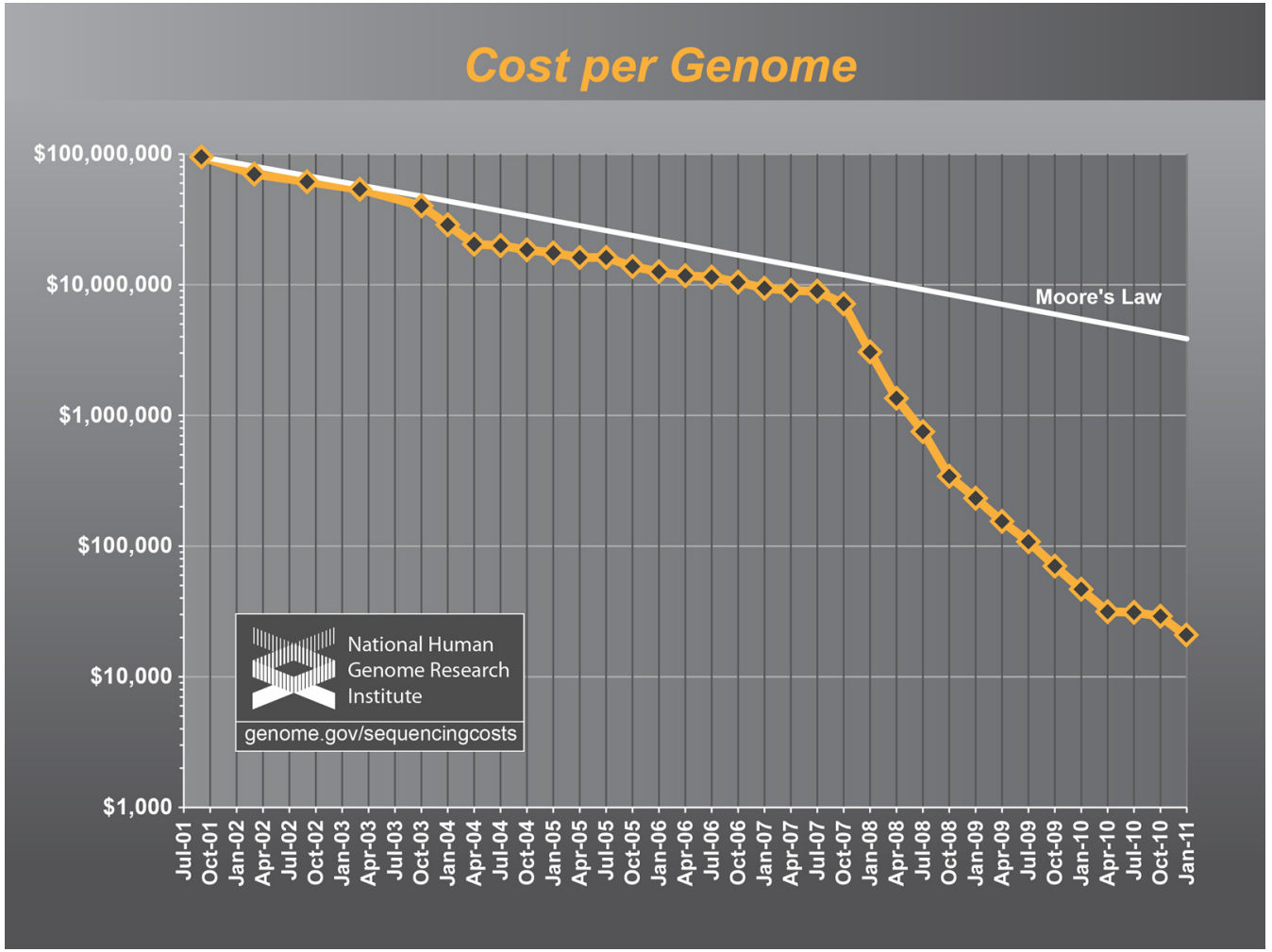
SAGA
Simple API for Grid Applications

Application Exemplar I: Ensemble and Replica-Exchange Simulations

- Replica-Exchange (RE) methods:
  - Represent a class of algorithms that involve a large number of loosely-coupled ensembles
  - Pattern not amenable to CIRRUS; explore Azure native abstractions
- RE simulations are used to understand a range of physical phenomena
  - Protein folding, unfolding etc
  - MC simulations
- Many successful implementations
  - e.g. folding@home [replica based]

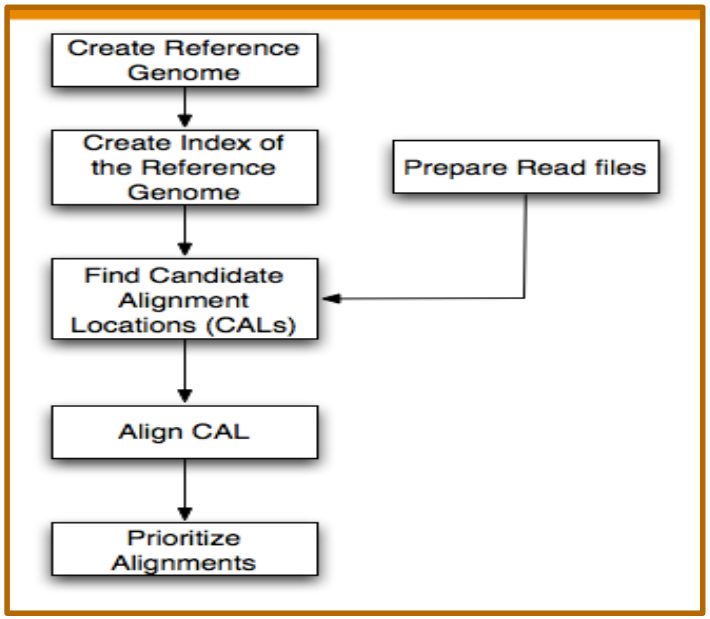


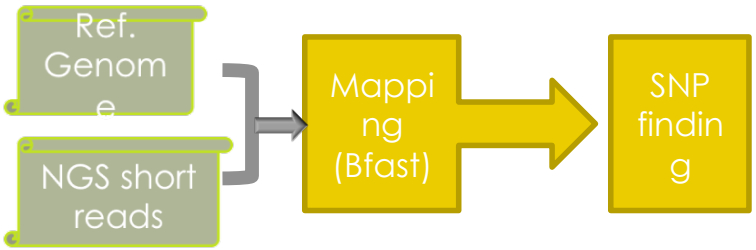
Application Exemplar II: NGS Analytics



BFAST: An example of NGS Analytics

- Higher sensitivity (CAL finding and gapped Smith-Waterman alignment)
- Relatively large memory and disk space
- Data types: (i) Short- Read (ii) Reference (iii) Index data
- Advanced features: (i) Multi-threading support (ii) Low-memory option (index file splitting)
- Breaking up short-read data permits task-level concurrency
  - Each Task requires full reference genome – possible I/O bottleneck
  - Distribute to over I/O bottleneck?
- Tradeoffs: Comp. vs Mem. vs I/O vs DoD
  - Sensitive to specific data-set size





SAGA
Simple API for Grid Applications

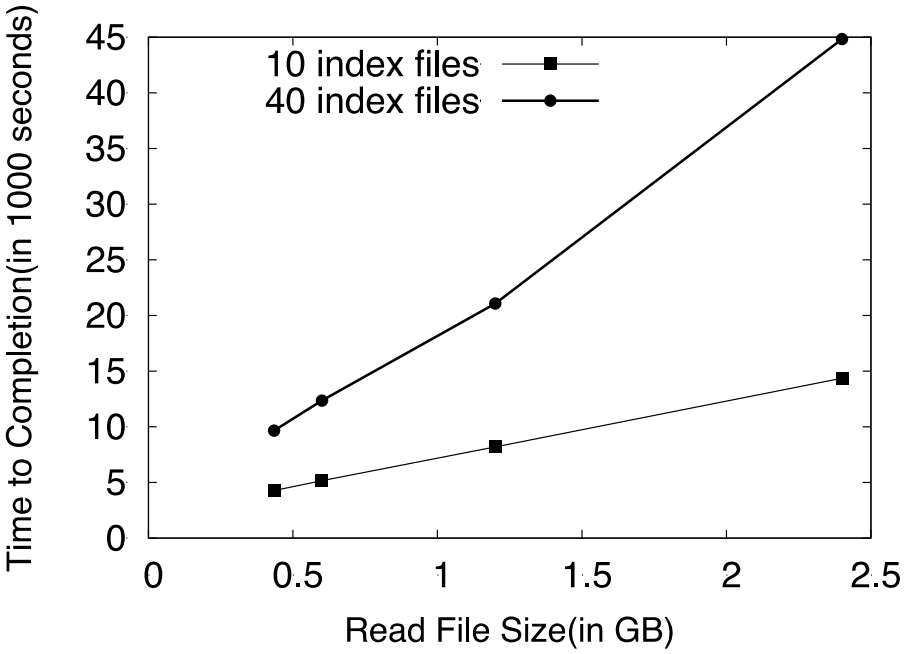
BFAST: Configuration values for problem sizes

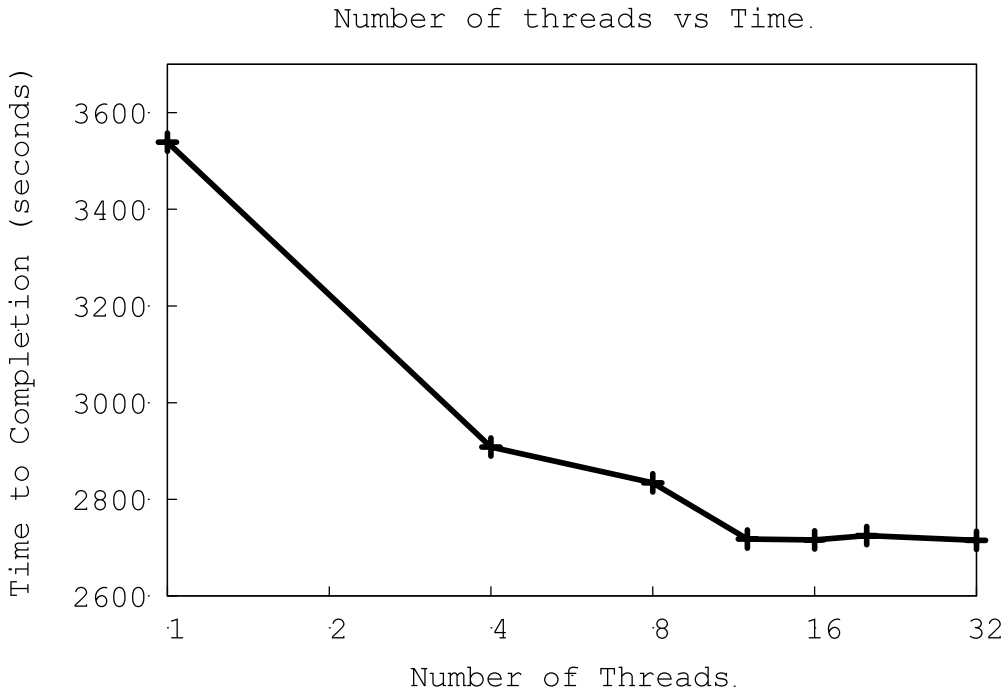
		B. Glumae (BG)	Human Genome (HG)	HG - Chr 21
Genome Size		7 Mbp	2.8Gbp (hg18)	47 Mbp (hg18-chr21)
NGS Type		Whole Genome	Exome	Exome
NGS Data Size		12.6 GB	5.6 GB	5.6 GB
Ref. Index Data	40 index files	40 GB	130 GB	70 GB
	10 index files			
Minimum Memory	40 index files	9.5 MB	3 GB	42 MB
	10 index files	38 MB	12 GB	164 MB

SAGA
Simple API for Grid Applications

BFAST Tradeoffs: Comp. vs Mem. vs I/O vs DoD

Possible solution: Logical and then physical distribution





Case	Read File Size	Threads per Task	# of Tasks	T_C
s1	2.4 GB	4	1	9358 s
s2	1.2 GB	2	2	5290 s
s3	0.6 GB	1	4	9152 s
s4	0.6 GB	4	4	5804 s

What are the Challenges for LS Applications on Clouds?

- LS Applications have
  - Memory vs I/O vs CPU vs DoD Tradeoffs
    - Multi-parametric trade-offs exist
  - "Complex" coordination requirements
- Distributed Applications Revisited
  - Where, when, how to distribute? How to manage coordination?
  - What is the task decomposition granularity? Mapping to resources?
  - What are the data transfer/access/storage mechanisms
- Need to support heterogeneous, distributed, dynamic loads
  - DA challenges need to be addressed dynamically!
    - Resource Elasticity/Cloudburst + Heterogeneous task-resource binding and need to for application configuration trade-offs
    - Models of localisation

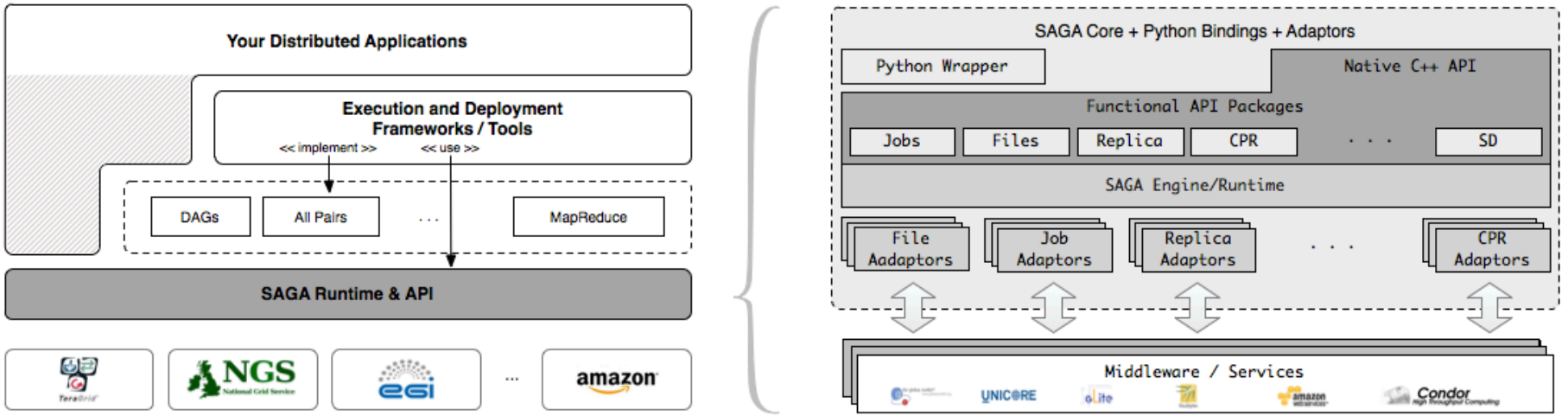
What are the Challenges for LS Applications on Clouds?

- What is the Cloud infrastructure configuration?
  - No well-defined single infrastructure configuration or capabilities
    - Contrast: Astronomy, HEP community
  - "TeraGrid is not used for data-intensive applications" (Fox)
- "Building this infrastructure is not trivial" (Fox)
  - Need Abstractions to Support Dynamic Applications
    - Both Development and System/Infrastructure level abstractions
  - There are "hard" parts and tractable parts
    - SAGA handles the hard part, opening up innovation elsewhere

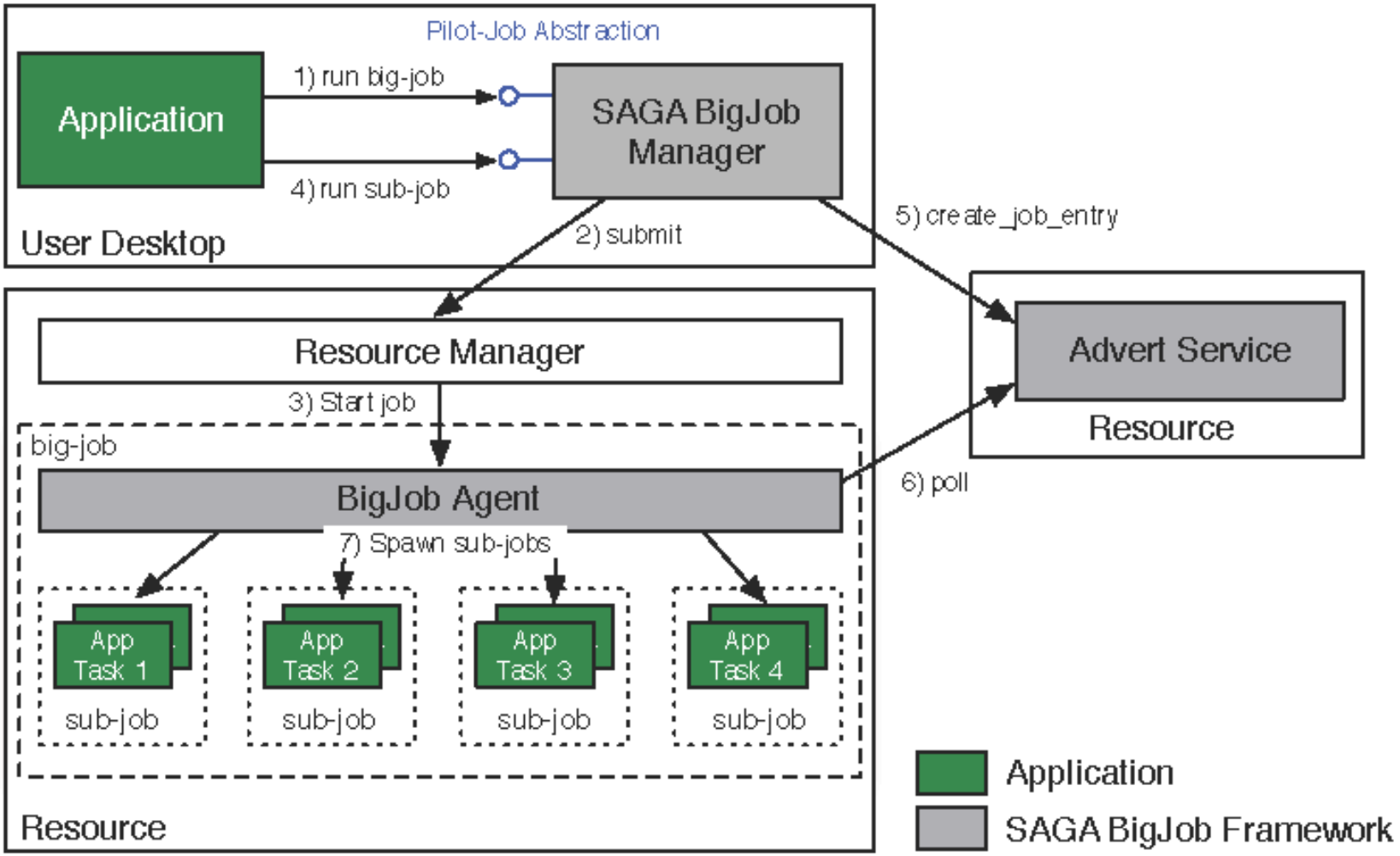
SAGA: In a nutshell

- There exists a lack of Programmatic approaches that:
  - Provide general-purpose, basic & common distributed functionality for applications; hide underlying complexity, varying semantics..
  - The building blocks upon which to construct "consistent" higher-levels of functionality and abstractions
  - Meets the need for a Broad Spectrum of Applications
    - Simple scripts, Gateways, Tooling, Workflow…
- Simple, integrated, stable, uniform and community-standard
  - Simple and Stable: 80:20 restricted scope
  - Integrated: Similar semantics & style across primary functional areas
  - Uniform: Same interface for different distributed systems
  - OGF-standard, "official" Access Layer/API of EGI, NSF-XD
- Standards-based approach: A Technical and an Economic (Moral?) imperative & case:

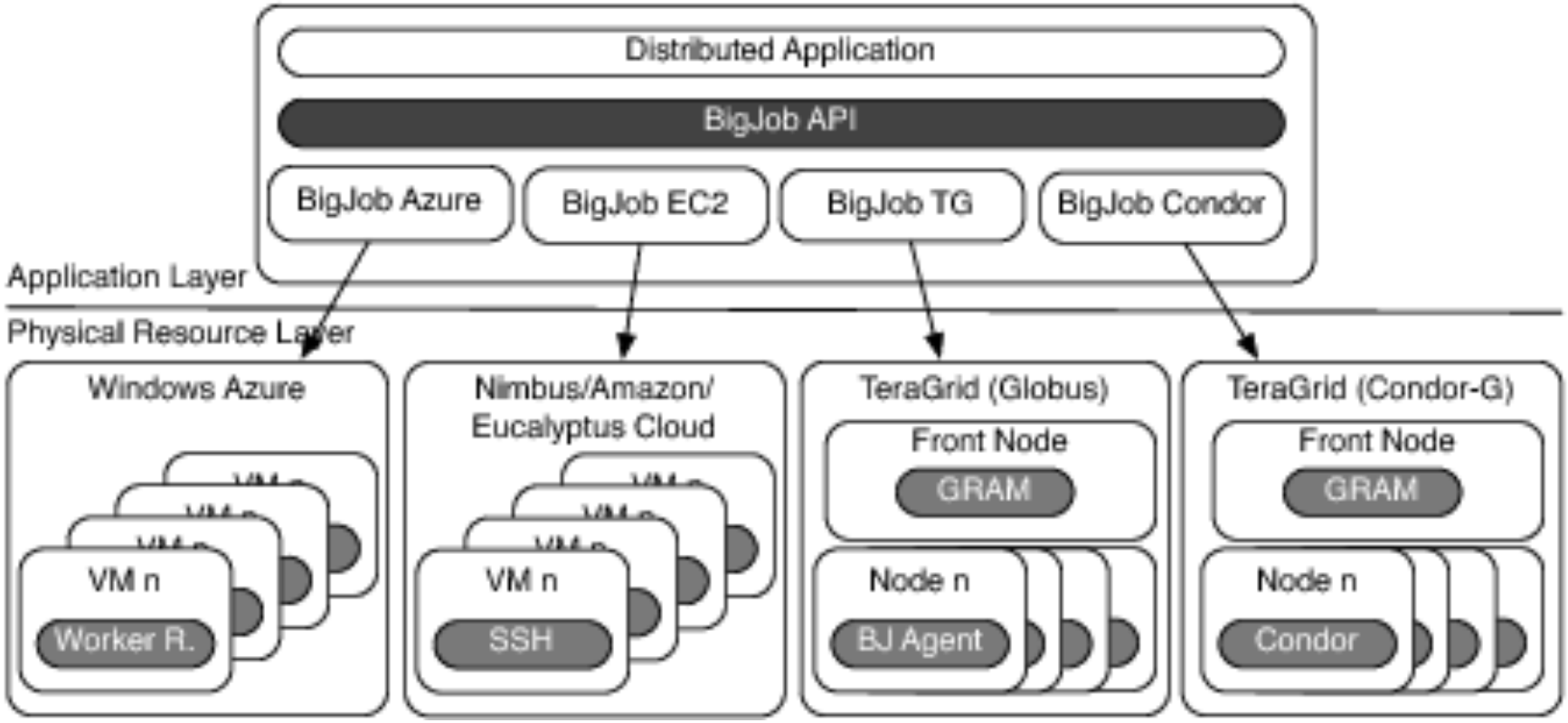
SAGA – An Overview



Abstractions for Dynamic Execution
SAGA Pilot-Job (BigJob)



Deployment & Scheduling of Multiple Infrastructure Independent Pilot-Jobs



SAGA
Simple API for Grid Applications

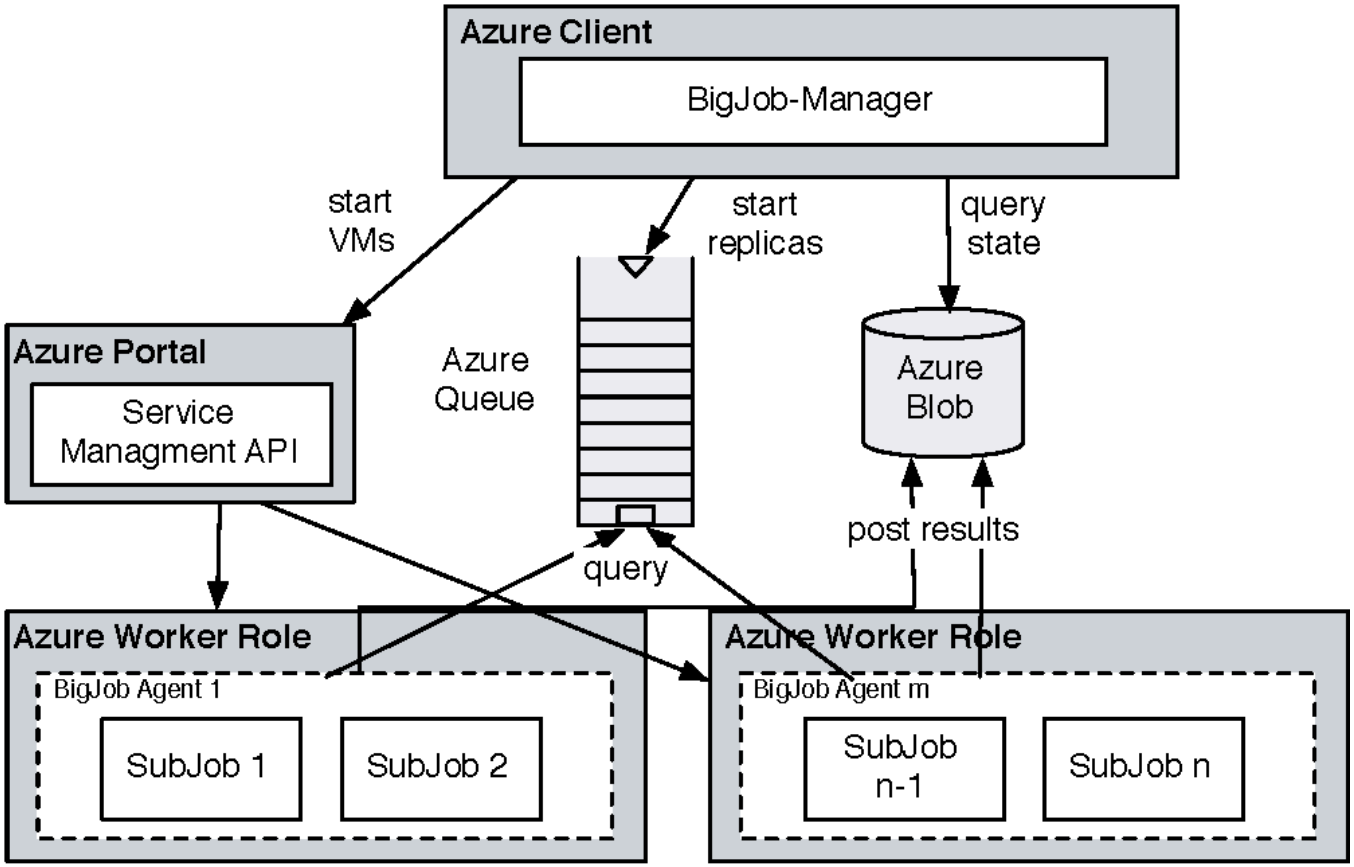
What is "unique" about Pilot-Jobs built using the right abstractions?

- Pilot-Jobs: Decouple resource assignment & resource-workload binding
- Pilot-Jobs are/have been typically used for:
  - Enhancing resource utilization; Facilitate high-throughput simulation
  - Lowering wait time for multiple jobs (better predictability)
- Several unique aspects about the SAGA-based Pilot-Job
  - Pilot-Jobs have not been used for Science Driven Objectives:
    - First demonstration of multi-physics simulations, REMD simulations
    - Frameworks based upon PJs (pull model) for specific back-ends
  - Infrastructure Independent and "standard" PJ API to access other PJs
    - SAGA PJ (BigJob) API basis for inter-operable PJ (Azure, DIANE)
- SAGA-based Pilot-Job form the basis for
  - Extension of Pilot-abstraction to other "dimensions"
  - For autonomic scheduling and application-level scheduling
  - Advanced run-time frameworks for load-balancing and FT

Ensemble MD simulations: BigJob for Azure

- Azure BigJob API calls Azure APIs directly
- BigJob Manager (BM) launches requested number of worker roles using the Service Management API
  - Python library created for this capability
- BigJob Agents run within the worker roles (C#/.Net)
  - Primary responsibility to execute MD code
  - MPI-based MD tasks confined to a single worker role (8 cores)
- For each sub-job, the BM creates a work-package
  - Distributed to agents using AQS – reliable and scalable msgs
  - Agents query AQS for new work-packages
  - Stage data from ABS

Coordinating Multiple Tasks Using BigJob for Azure



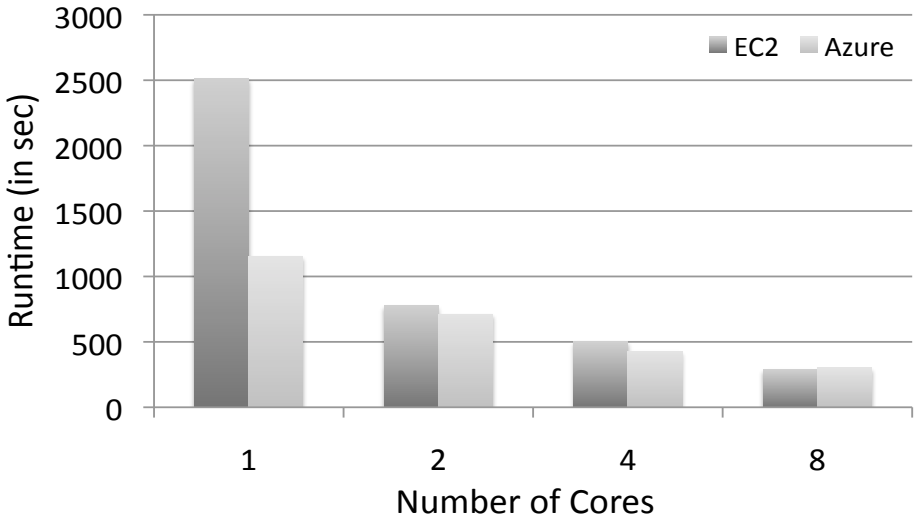
- Multiple Affinity Groups supported within the same BigJob for different worker-roles

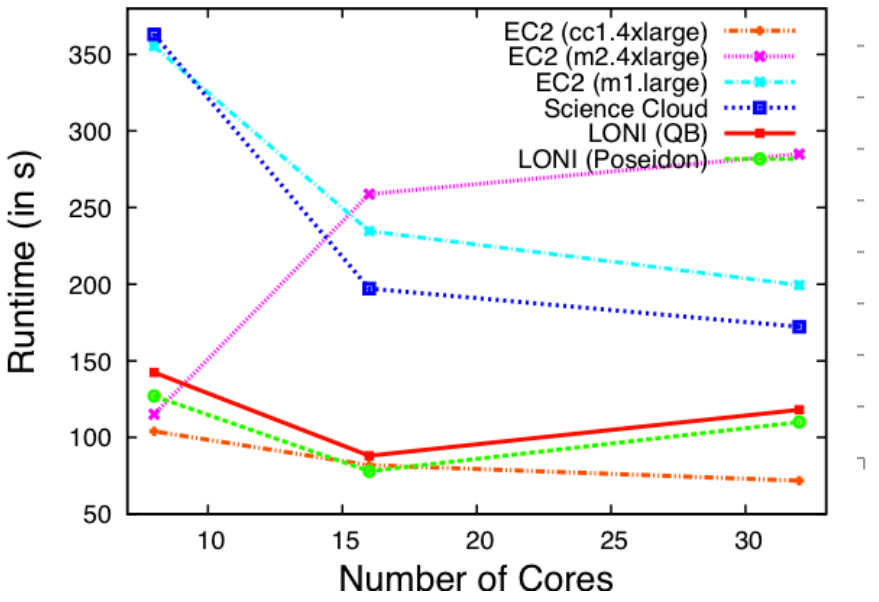
SAGA
Simple API for Grid Applications

Azure: Scalability with Simplicity
Providing Infra-level abstractions for DDIA

- Three types of storage abstractions:
  - Azure Blob Service: Large amounts of raw data
    - Block Blob: Large chunks of 5GB
    - Page Blob: Mange storage as an array
  - Azure Table Storage: Semi-structured data
  - Azure Queue Storage: Message Queues
  - All three replicated and with strong consistency semantics
- Current affinity operates at the data-center level;
  - Enhanced/finer-grained affinity to be made available

NAMD Performance on Azure

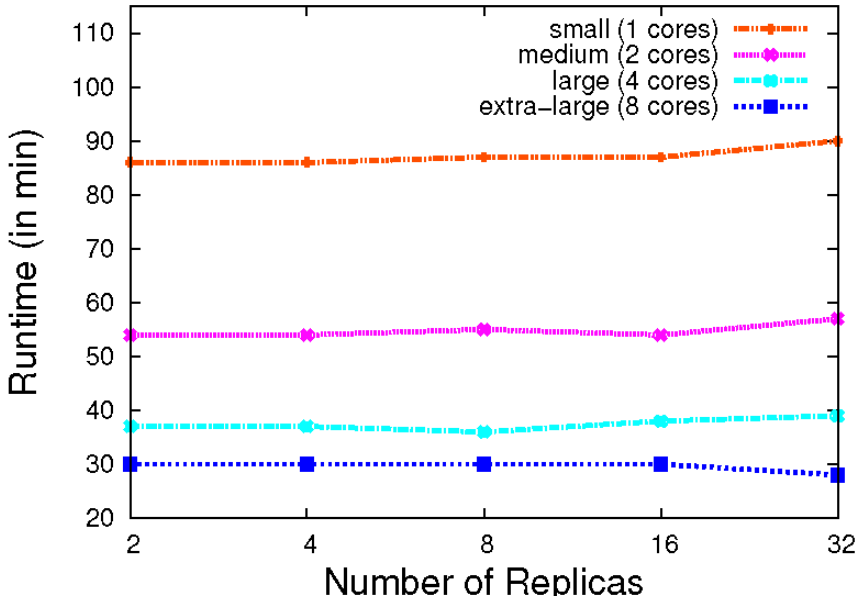


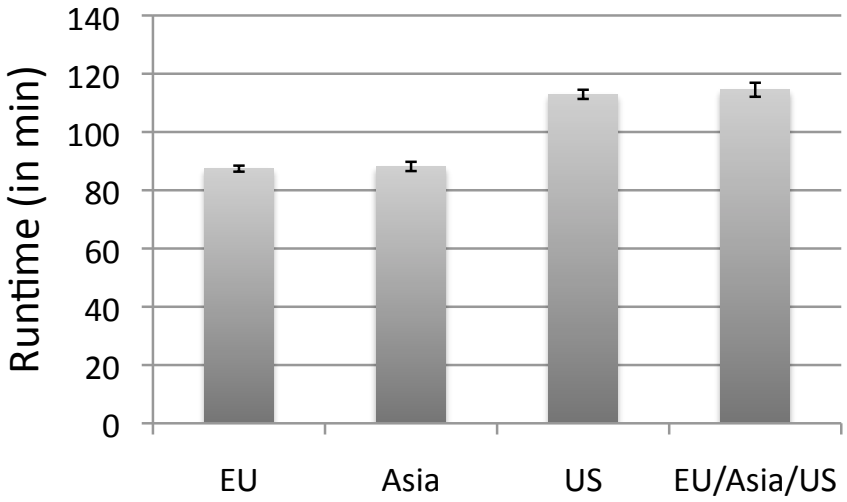


SAGA
Simple API for Grid Applications

Replica-Exchange on Azure

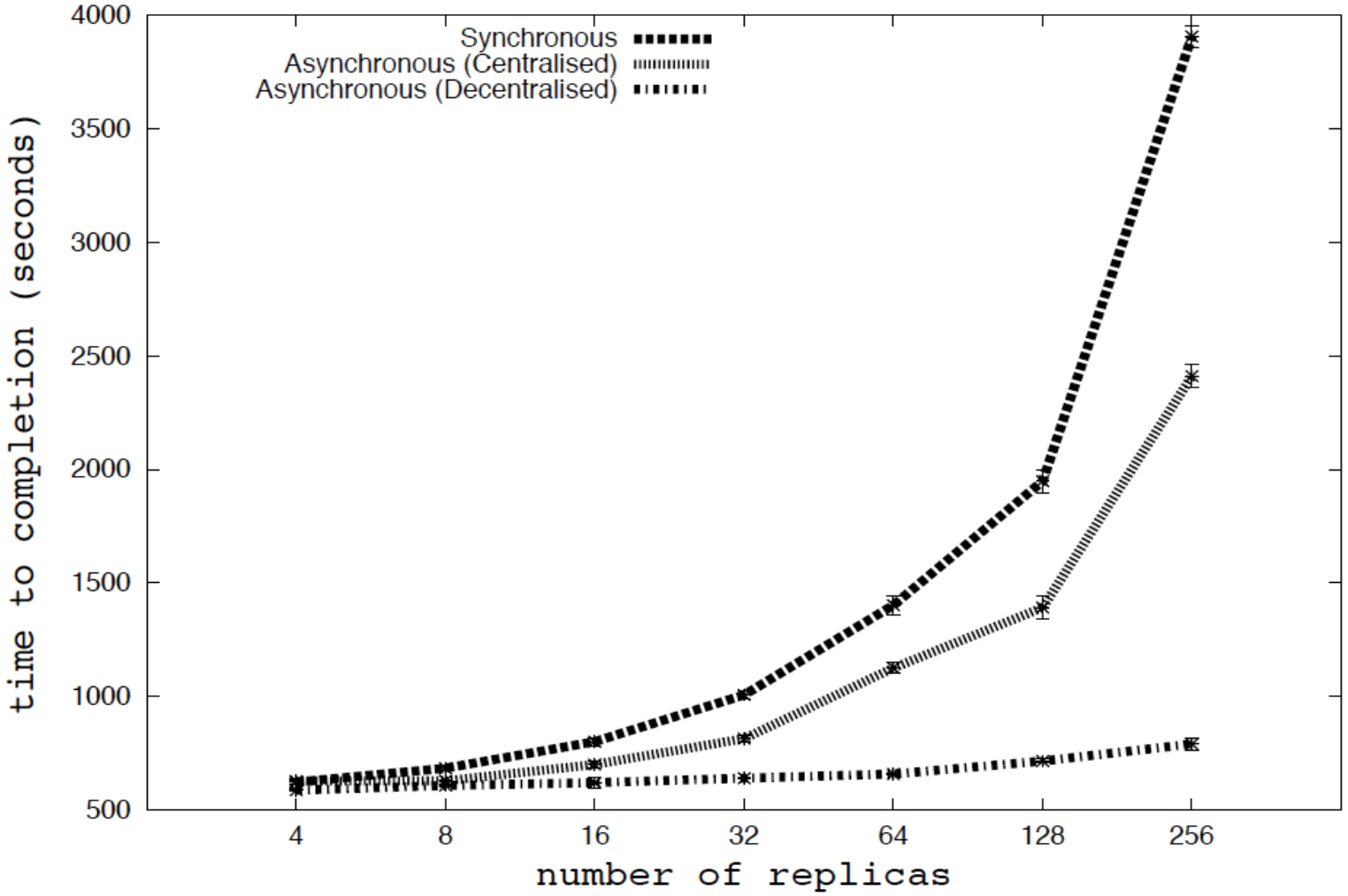
- Azure BigJob scales well with the number of replicas.
  - AQS proved to be effective for coordination of sub-jobs/replicas
- Larger VMs have a better performance.
  - But, efficiency drops < 0.4 for the extra-large VM
- The different Azure data centers show a slight fluctuation in their performance
  - 16 replicas, small VM



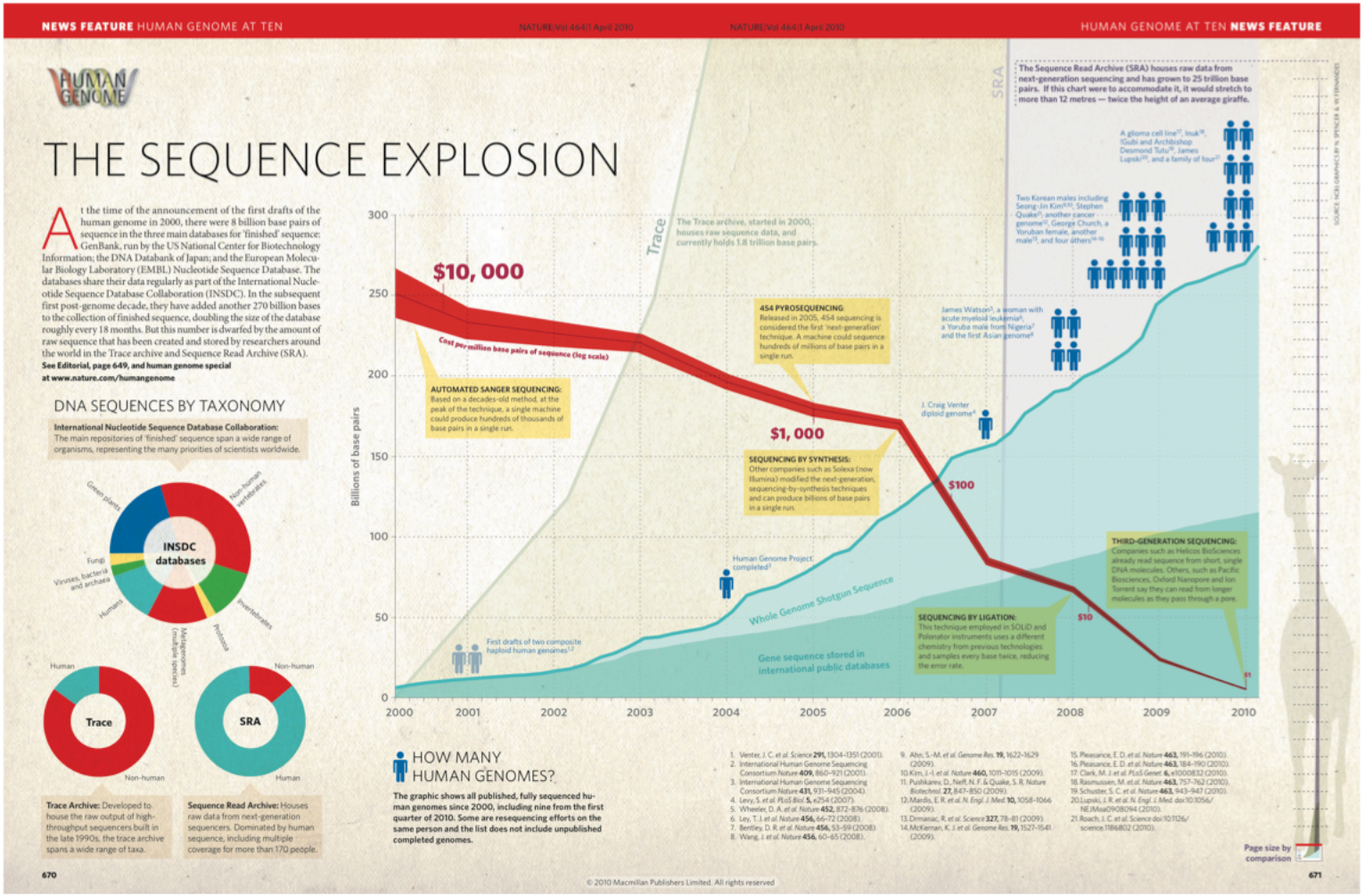


SAGA
Simple API for Grid Applications

RE Algorithms at Scale (TeraGrid)
Replicate/Understand Algorithms at Scale on Azure?

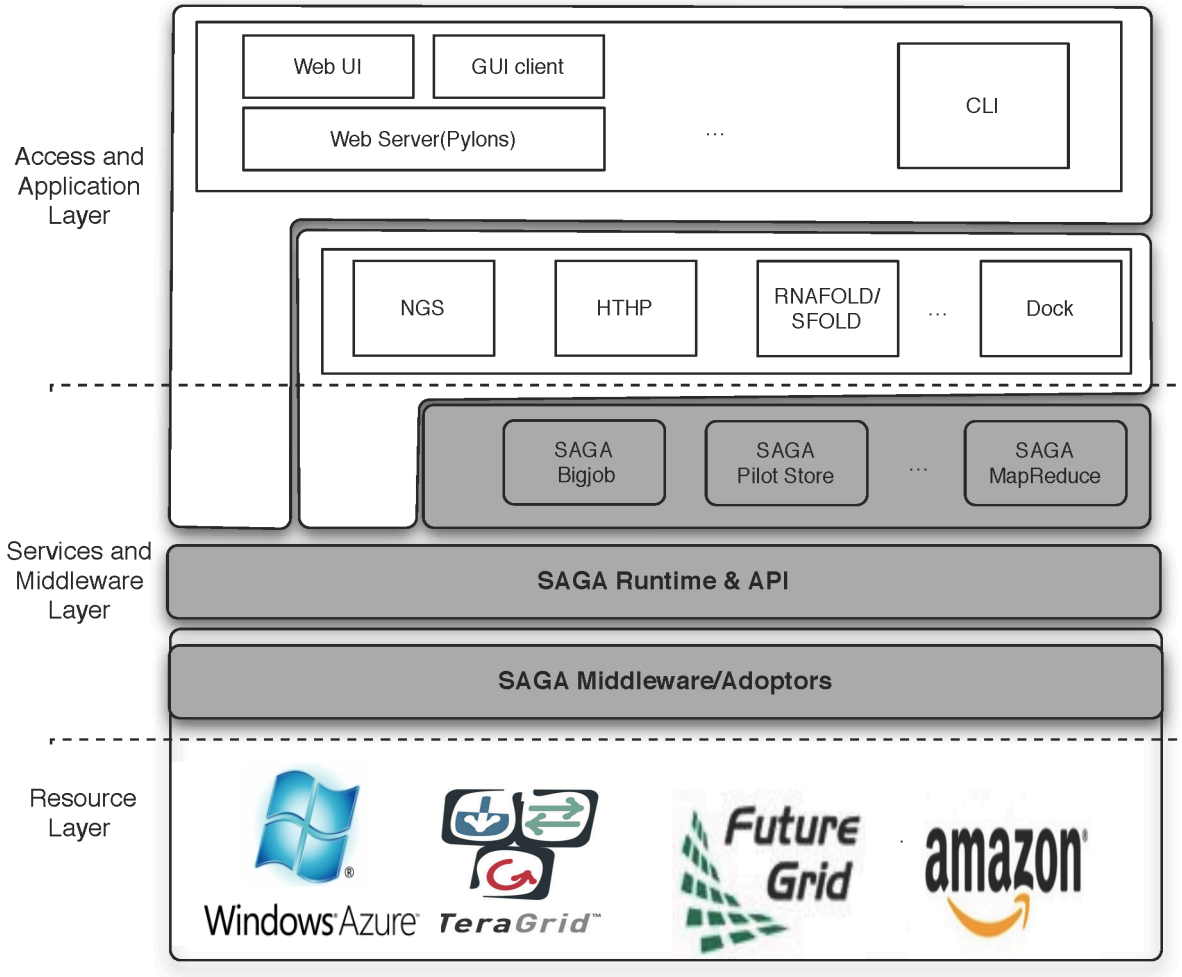


Application Exemplar II: NGS Analytics



DARE-based Science Gateways

- Provides fundamental abstractions for Dynamic and Adaptive Execution
  - Integrated compute and data
- Efficient and novel runtime environments for Map-Reduce
- Interoperable across DCI
- Extensible: new features and abstractions



Tradeoffs: Comp. vs Mem. vs I/O vs DoD

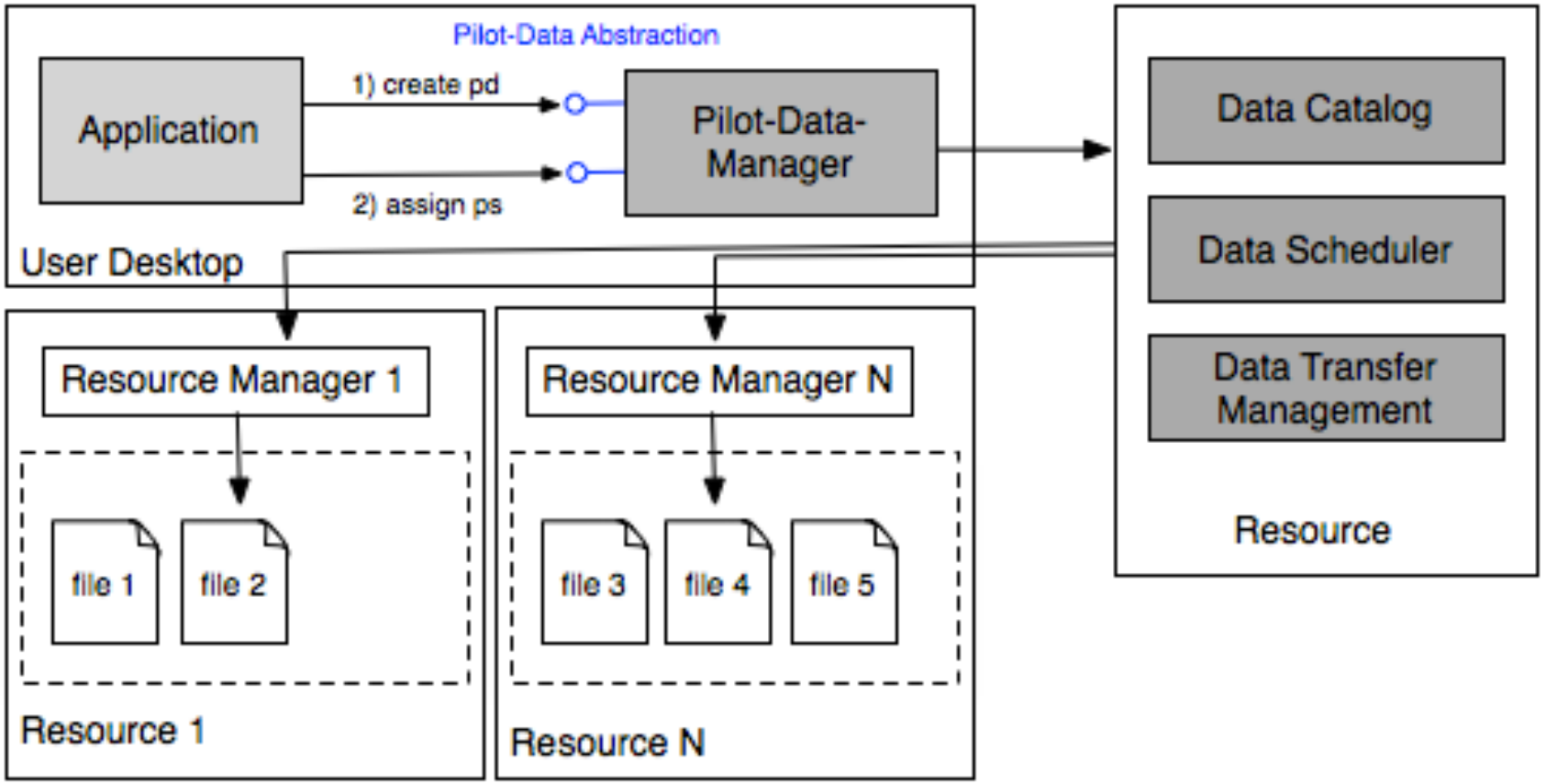
Genome Type	Index File (GB)	Resource	# of Cores	# of nodes	# of VMs	TTC (sec)
BG	0.435	R	64	4	-	1067
BG	0.435	QB	64	8	-	719
BG	0.435	R/QB	32/32	2/4	-	919
BG	0.435	FG	64	-	8	712
BG	0.435	R/QB/ FG	24/24/ 24	2/3	3	1022
Chr	1.9	R	64	4	-	1145
Chr	1.9	QB	64	8	-	924
Chr	1.9	R/QB	32/32	4/2	-	1170
HG	127	R	256	16	-	9586
HG	127	R/QB	128/128	8/16	-	7582

Extending the Pilot-Abstraction to Data
Pilot-Data Features

Ack: Initial discussions with Jon Weissman

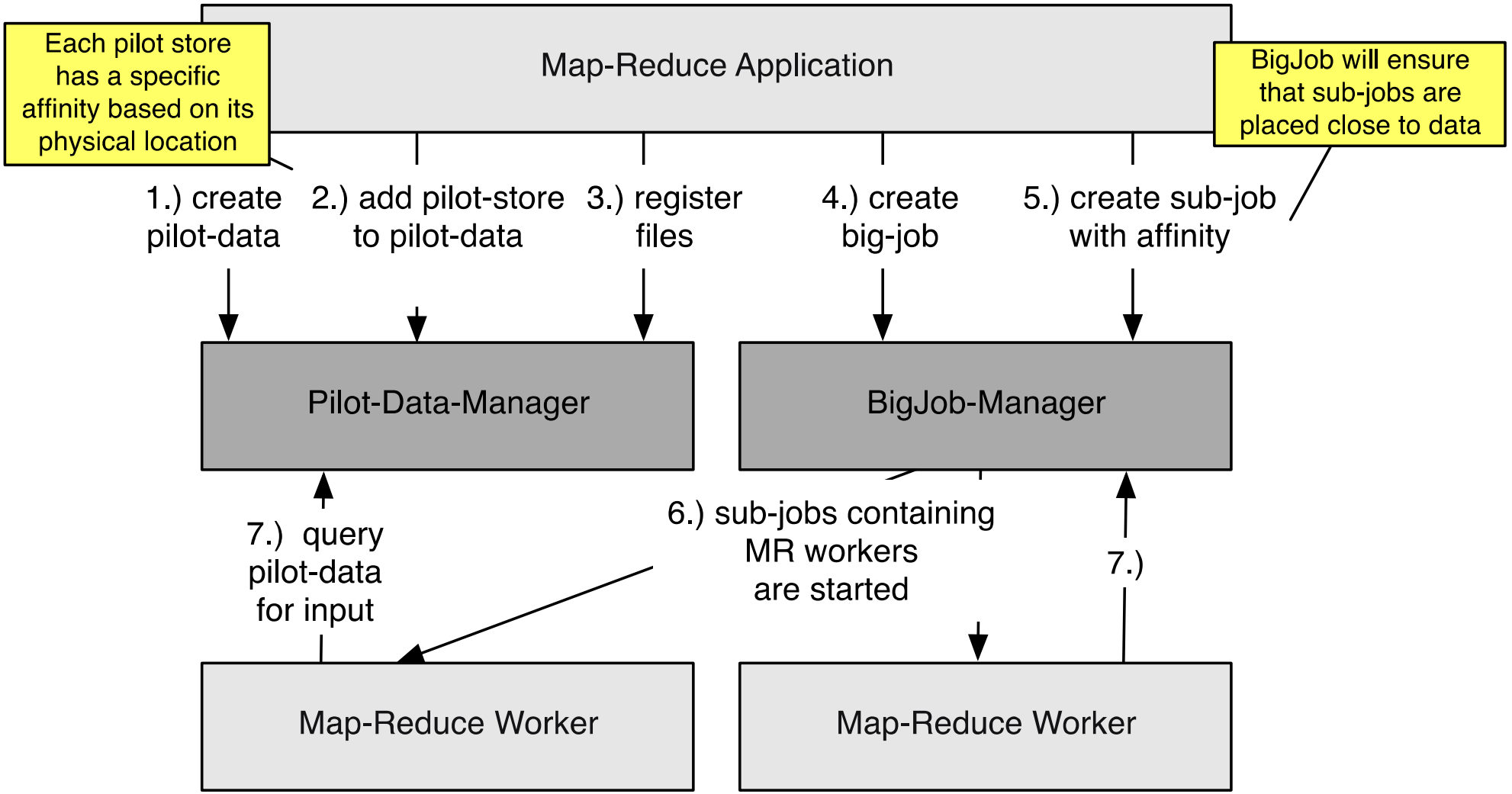
- Expression of affinities between file groups (data-data) as well as files and compute resources (data-compute)
  - Grouping of files that are often used together
  - Data partitioning and distribution of files
- Distributed access and file movement
- Integration with Pilot-Job
  - For data-aware scheduling
  - Supporting and Implementing affinity at middleware/tool level
- Future Work
  - Data scheduling
  - Data replication and consistency management
  - Integration of third party data management frameworks

Extending the Pilot-Abstraction to Data



SAGA
Simple API for Grid Applications

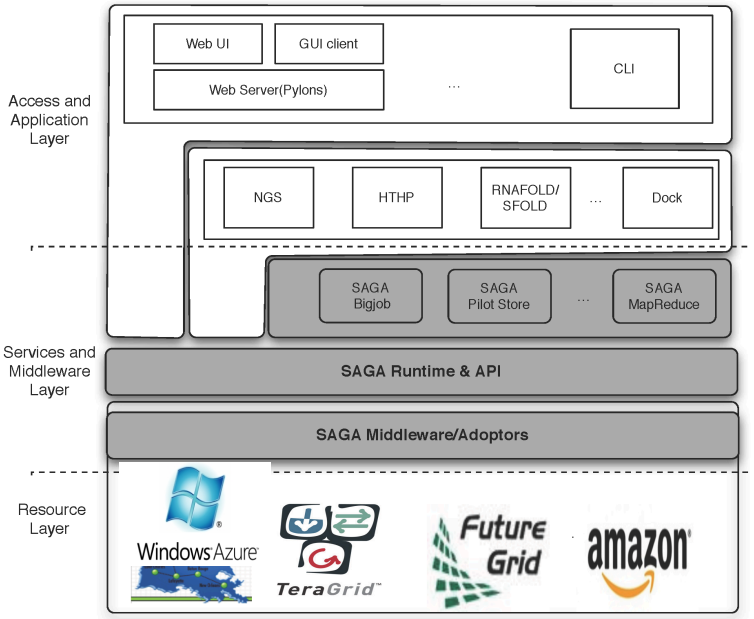
Map-Reduce using Pilot-Abstractions



SAGA
Simple API for Grid Applications

Towards NGS Analytics as a Service: DARE-based Gateway on XD

- Some NGS specific challenges
  - Efficient Algorithm/tool/code selection
    - Hosting pre-installed VM
  - Efficient task scheduling and placement
    - What can we learn from HEP? WMS?
  - Efficient Distributed data management
  - Efficient Data transfer/scheduling
    - Transfer of Ref. genome index files:
      - O(hours) 130 GB,
    - Transfer of Short read files:
      - O(mins) [L to QB] 9 GB
  - Determine optimal point -- tradeoff



http://dare.cct.lsu.edu/gateways/ngs

- Solutions Applicable to Azure
  - Stand-up a NGS service for Azure community?
  - Extensible to other "services" for drug-discovery and bioinformatics
    - http://dare.cct.lsu.edu/

Conclusions

- LS Applications – compute and data intensive present broad range of challenges at scale
- Combination of appropriate system-level abstractions (e.g. AQS) and user provided abstractions (e.g., BigJob)
  - Relative Ease of implementation of the R-E Pattern
    - Efficient and scalable messaging
  - Performance comparable to TG
    - Cost of virtualization not a first order concern
- Ready/Need for advanced abstractions + implementation
  - Extensions to abstractions for dynamic data
  - Affinity becomes more fine-grained, data-compute affinity
    - E.g. map affinity in PS/PJ to Azure

Acknowledgements

SAGA Team and contributors: http://saga.cct.lsu.edu

Futuregrid Acknowledgement

This document was developed with support from the National Science Foundation (NSF) under Grant No.0910812 to Indiana University for "FutureGrid: An Experimental, High-Performance Grid Test-bed." Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reaect the views

Also Acknowledge useful discussions:

Geoffrey Fox and Jon Weissman