Developing Data-Intensive Applications for Heterogeneous Distributed Platforms

Shantenu Jha

SAGA: C Miceli, M Miceli, A Merzky S Sehgal, M Erdelyim, K Stamou
DIC: Dan Katz (Univ of Chicago), Jon Weissman (Univ of Minnesota)

http://saga.cct.lsu.edu

Outline

- The Case for Distributed DI Application
- Critical Perspective on Distributed Application Development
- **IDEAS**: First principles design objectives for Distributed App
- SAGA – Promoting IDEAS
- SAGA-based Applications
  - Four Applications
    - How IDEAS are met for these applications
    - Increasing in irregularity/structure
  - Understanding the landscape through examples
  - Aim: To show the kind of questions that need asking

Case for Developing Extensible DDIA

- Data inherently distributed
  - Distributed DIA, not just the simple sum of DIA concerns
- Multiple, Heterogeneous Infrastructure
  - Decouple Application Development from underlying infrastructure
    - Scale-out (and not just Scale-up)
  - Interoperation, e.g., concurrently cross Grid-Clouds
- Support Runtime or Application Characteristics for multiple applications and different infrastructure
  - Support Multiple Programming Models
    - Master-Worker, but Irregular versus Regular Workload
  - Support Application-Level Patterns
    - MapReduce, File-based versus Stream-based
  - Support Distributed Affinities

Distributed Data Intensive Applications
Research Challenges

- Goal: Develop DDI scientific applications to utilize a broad range of distributed systems, without vendor lock-in, or disruption, yet with the flexibility and performance that scientific applications demand.
  - Frameworks as possible solutions
- Frameworks address some primary challenges in developing Distributed DI Applications
  - Coordination of distributed data & computing
  - Runtime (Dynamic) scheduling, placement
  - Fault-tolerance
- Many Challenges in developing such Frameworks:
  - What are the components? How are they coupled? Functionality expressed/exposed? Coordination?
  - Layering, Ordering, Encapsulations of Components

Distributed Applications

Critical Perspectives

- Details at: http://grid2009.org/bestpaper
- Ability to develop simple, novel or effective distributed applications lags behind other aspects of CI
  - Distributed CI: Is the whole > than the sum of the parts?
- Infrastructure capabilities (tools, programming systems) and policy determine applications, type development & execution:
  - Proportion of App. that utilize multiple distributed sites sequentially, concurrently or asynchronously is low
    - Not referring to tightly-coupled across multiple-sites
  - Focus on extending legacy, static execution models
    - Scale-Out of Simulations? Compute where the data is?
- What novel applications & science has Distributed CI fostered
  - Distinguish challenges of provisioning Distributed CI versus support for application development

Critical Perspectives
Quick Analysis

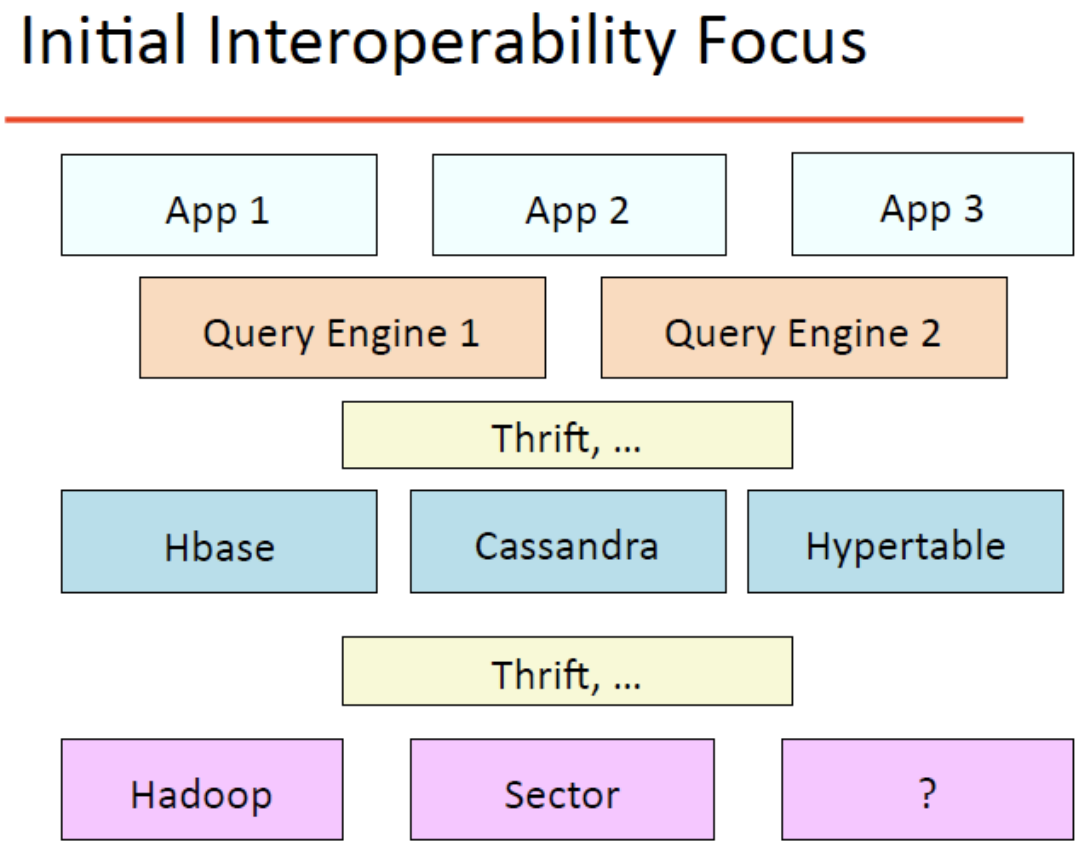
- Several Factors responsible for perceived & actual lack of DA
  - Developing Distributed Applications is fundamentally hard!
    - Coordination across multiple distinct resources
  - Range of tools, prog. systems and environments large
    - Interoperability and extensibility become difficult
  - Commonly accepted abstractions not available
    - E.g. Pilot-Job powerful, but no "unifying" tool on TG
  - Deployment and execution capabilities disjoint from the development concerns/process
    - Generally good idea, but application development often influences where and how it can be deployed/executed

Understanding Distributed Applications
IDEAS: First Principles Development Objectives

- **Interoperability:** Ability to work across multiple distributed resources
- **Distributed Scale-Out:** The ability to utilize multiple distributed resources concurrently
- **Extensibility:** Support new patterns/abstractions, different programming systems, functionality & Infrastructure
- **Adaptivity:** Response to fluctuations in dynamic resource and availability of dynamic data
- **Simplicity:** Accommodate above distributed concerns at different levels easily…

Challenge: How to develop DA effectively and efficiently with the above as first-class objectives?

Interoperabilty: Motivation



Slide adapted: Grossman

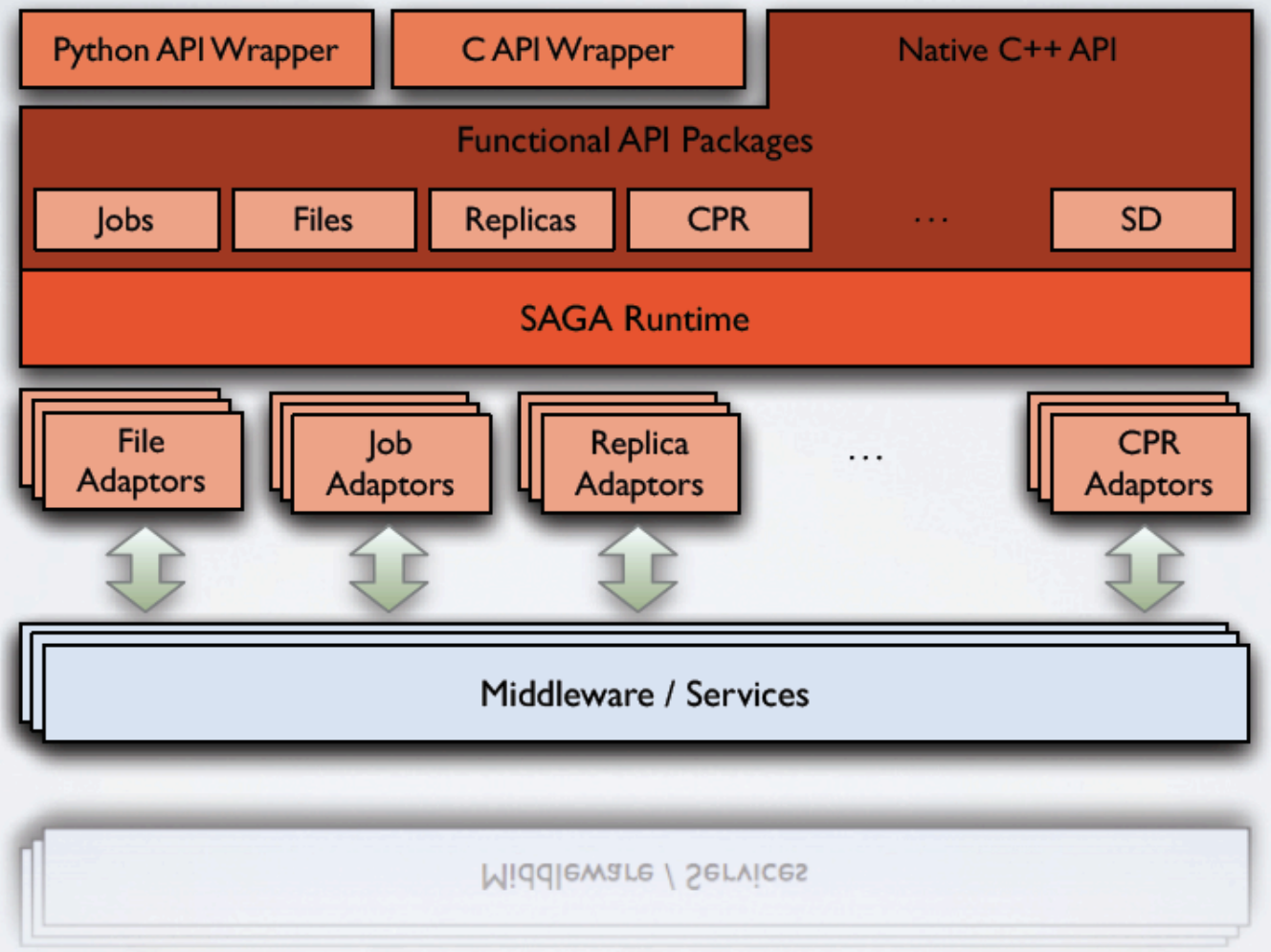
Decouple from vertical stack
Decouple from details of resource provisioning
Couple horizontal stacks

SAGA: In a nutshell

- There exists a lack of Programmatic approaches that:
  - Provide general-purpose, basic &common grid functionality for applications and thus 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 Application:
    - Simple scripts, Gateways, Smart Applications and Production Grade Tooling, Workflow…
- Simple, integrated, stable, uniform and high-level interface
  - Simple and Stable: 80:20 restricted scope and **Standard**
  - Integrated: Similar semantics & style across
  - Uniform: Same interface for different distributed systems
- SAGA: Provides Application* developers with units required to compose high-level functionality across (distinct) distributed systems

  (*) One Person's Application is another Person's Tool

SAGA: In a thousand words..



SAGA C++ quick tour

- Open Source - released under the Boost Software License 1.0
- Implemented as a set of libraries
  - SAGA Core - A light-weight engine / runtime that dispatches calls from the API to the appropriate middle-ware adaptors
  - SAGA functional packages - Groups of API calls for: jobs, files, service discovery, advert services, RPC, replicas, CPR, ... (extensible)
  - SAGA language wrappers - Thin Python and C layers on top of the native C++ API
  - SAGA middleware adaptors - Take care of the API call execution on the middleware
- Can be configured / packaged to suit your individual needs!

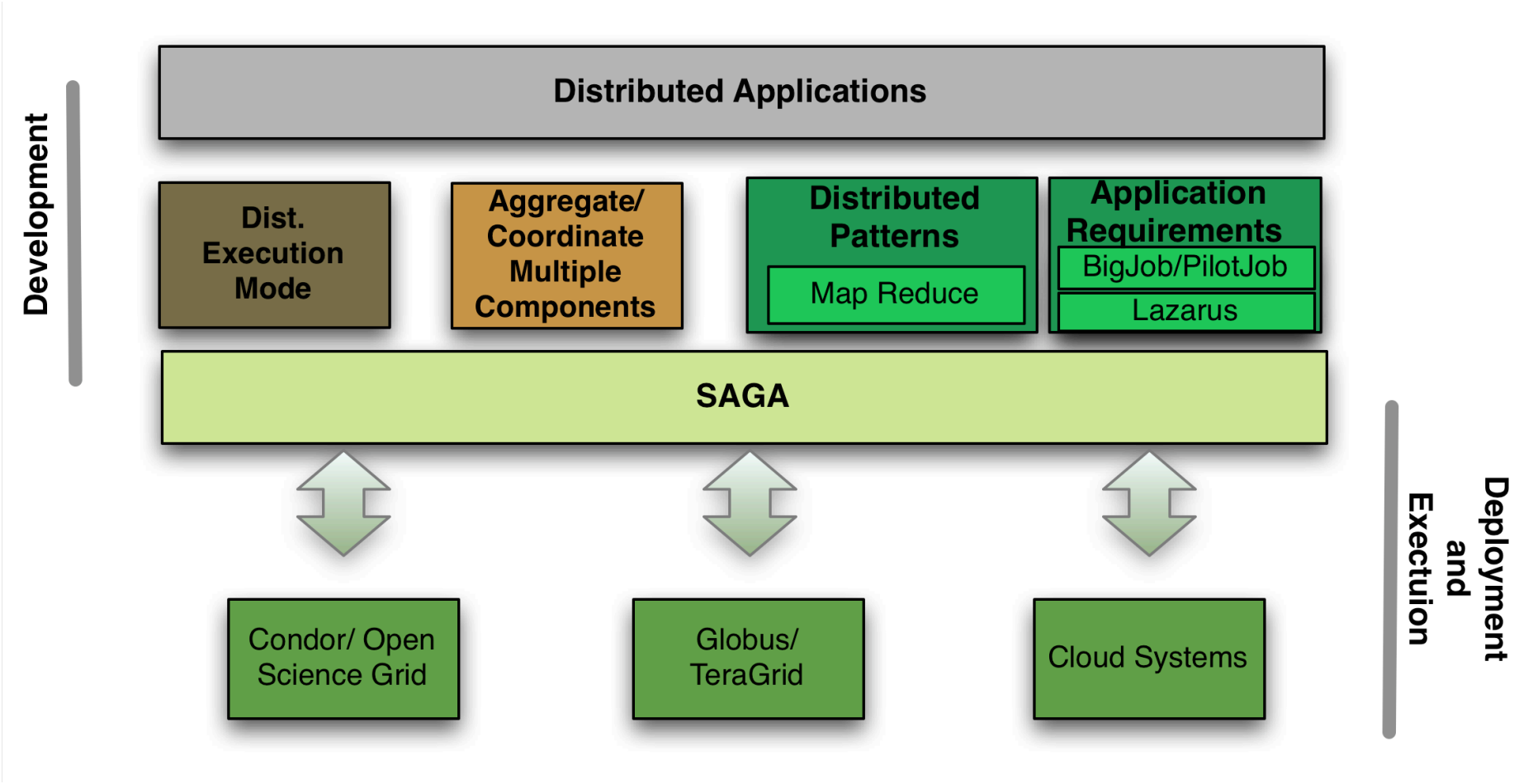
SAGA Implementation: Extensibility

- Horizontal Extensibility – API Packages
  - Current packages:
    - file management, job management, remote procedure calls, replica management, data streaming
    - Steering, information services, checkpoint…
- Vertical Extensibility – Middleware Bindings
  - Different adaptors for different middleware
  - Set of 'local' adaptors
- Extensibility for Optimization and Features
  - Bulk optimization, modular design

SAGA: Available Adaptors

- Job Adaptors
  - Fork (localhost), SSH, Condor, Globus GRAM2, OMII GridSAM, Amazon EC2, Platform LSF
- File Adaptors
  - Local FS, Globus GridFTP, Hadoop Distributed Filesystem (HDFS), CloudStore KFS, OpenCloud Sector-Sphere
- Replica Adaptors
  - PostgreSQL/SQLite3, Globus RLS
- Advert Adaptors
  - PostgreSQL/SQLite3, Hadoop H-Base, Hypertable

SAGA and Distributed Applications



SAGA-based frameworks: Logical ordering

Distributed Data Intensive Applications
e.g. Particle Physics, Astronomy, Bio-Informatics

AF

Programming Abstractions/Patterns/Higher-level APIs
e.g. MapReduce, All-Pairs, Scatter-Gather

RF

Common Runtime Support
e.g. Affinity, Fault Tolerance, Patterns

Physical Infrastructure
e.g. TeraGrid/XD, Clouds, Future Data Systems

SAGA-based Applications: Examples

- SAGA NxM Framework (All-Pairs)
  - Compute Matrix Elements, each is a Task
    - All-to-All Sequence comparison
  - Control the distribution of Tasks and Data
  - Data-locality optimization via external (runtime) module
- SAGA MapReduce Framework:
  - Control the distribution of Tasks (workers)
  - Master-Worker: File-Based &/or Stream-Based
  - Data-locality optimization using SAGA's replica API
- SAGA-based Sphere (Stream based processing)
- SAGA-based DAG Execution
  - Extend to support Load-balancing and dynamic decision/placement & scheduling
- Applications ordered from more to less regular
  - All-Pairs very structured C, D
  - DAG-based applications can be very irregular

SAGA-based All-Pairs

- We use a SAGA-based All-Pairs abstraction
  - Applies an operation on the input data-set such that every possible pair in the set is input to the operation
  - Degrees of freedom: Data assignment, Distribution
- Our application compares 'genome' (files) that consist of random combinations of ACGT
  - Other Examples: AP for Image Similarity (Biometrics) [D Thain]
- The application spawns (distributed) jobs to run sets of these pairs
  - Determining which pairs to put into a set, and on which distributed resource to run that set
  - Data distribution vs. computation
    - Data distribution/transfer times relevant

SAGA-based All-Pairs
Multiple-levels of "control"

- Initial Data Condition:
  - Data maybe distributed across resources, possibly localized
- Work Decomposition:
  - Granularity of work-load,
    - 8x8 matrix (=64 matrix elements): workload unit 4? 16? 64?
  - Workload to worker mapping
    - For a fixed data set size, this is equal to number of workers
  - Worker placement
    - All local? All distributed?
      - I/O saturation? Compute-bound? Network effects?
- Stage at which workload to workers binding takes place

DDIA: Some questions
SAGA-based All-Pairs

- We want to understand:
  - Performance sensitivity to data decomposition, workload granularity and distribution
  - Which infrastructure to use, for specific problem (data access patterns), or in general for a given application?
    - Performance tradeoffs of a DFS compared to "regular" distribution
    - Examine sensitivity to placement techniques

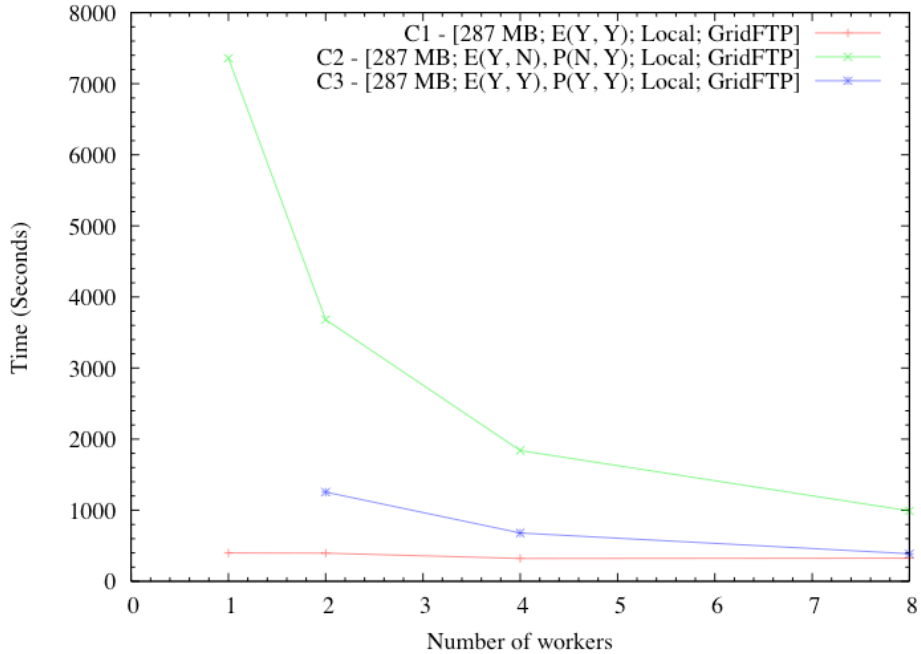
- Why DFS?
  - Abstract layer between application and local file systems
    - Simplified Handling Data
      - Handles data distribution (management)
      - Provides replication and other capabilities
  - Some examples include
    - HDFS – Hadoop's filesystem, GFS
    - CloudStore an open-source high performance DFS based on Google's distributed filesystem GFS.
  - Common to load DFS as part of VM/Image
  - Multiple (Open-Source) now available; generally more reliable now

To DFS, or not to DFS?

- Pros
  - Handles data distribution (management)
  - Provides Replication
  - Fault tolerance
  - Capability to handle data dependencies
- Cons
  - Overhead
  - Inability to control work placement
  - They all work differently. Not easy to understand performance!
- Performance:
  - Performance advantage of a DFS? What constraints?
  - Customize a DFS for further performance advantage?

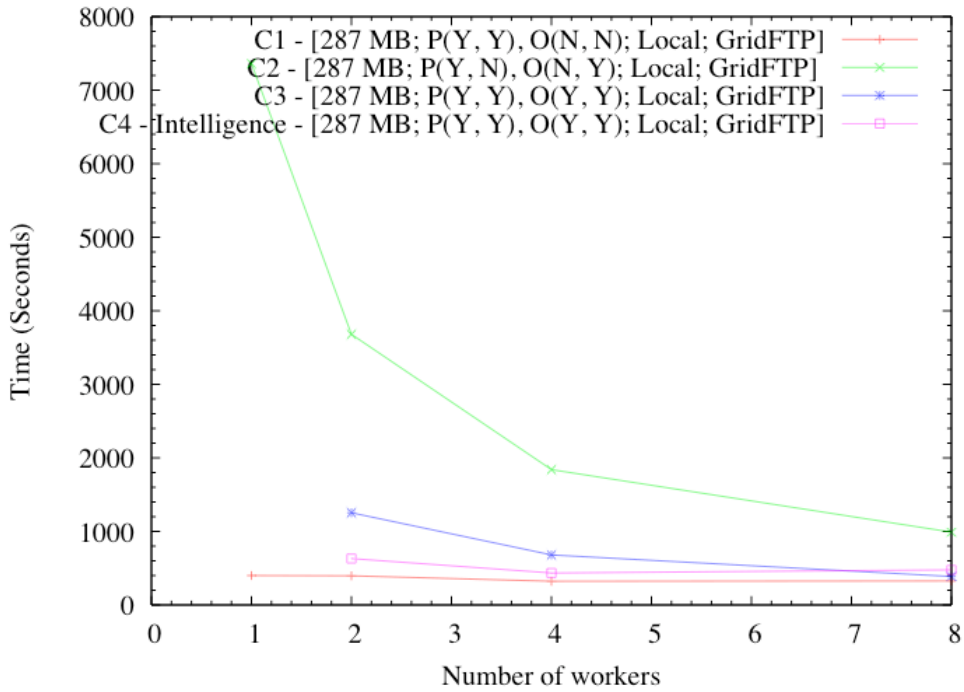
Distributed Data – Base Line tests

- Workload: 2.3GB
- Each file 287MB, thus 8x8 matrix.
- Configurations:
  - C1 = Local
  - C2 = C on R1; D on R2
  - C3 = C on R1, R2; D on R1,R2
- Time curves down, as N_W up
- Adding workers eventually becomes ineffective
  - Coordination costs dominate
- Accessing Remote data is expensive



Distributed Model (Intelligent)

- Workload 2.3GB
- Each file 287MB; thus 8x8 matrix
- Configurations:
  - C4 = C3 + Intelligence
- Very simple Intelligence: assign tasks upon lowest transfer time
- Intelligence Overhead negligible
  - Implementing Intelligence ~1% time
- Different file sizes
  - Scales similarly
- Scale out to > 2 resources (4)



Number of Workers

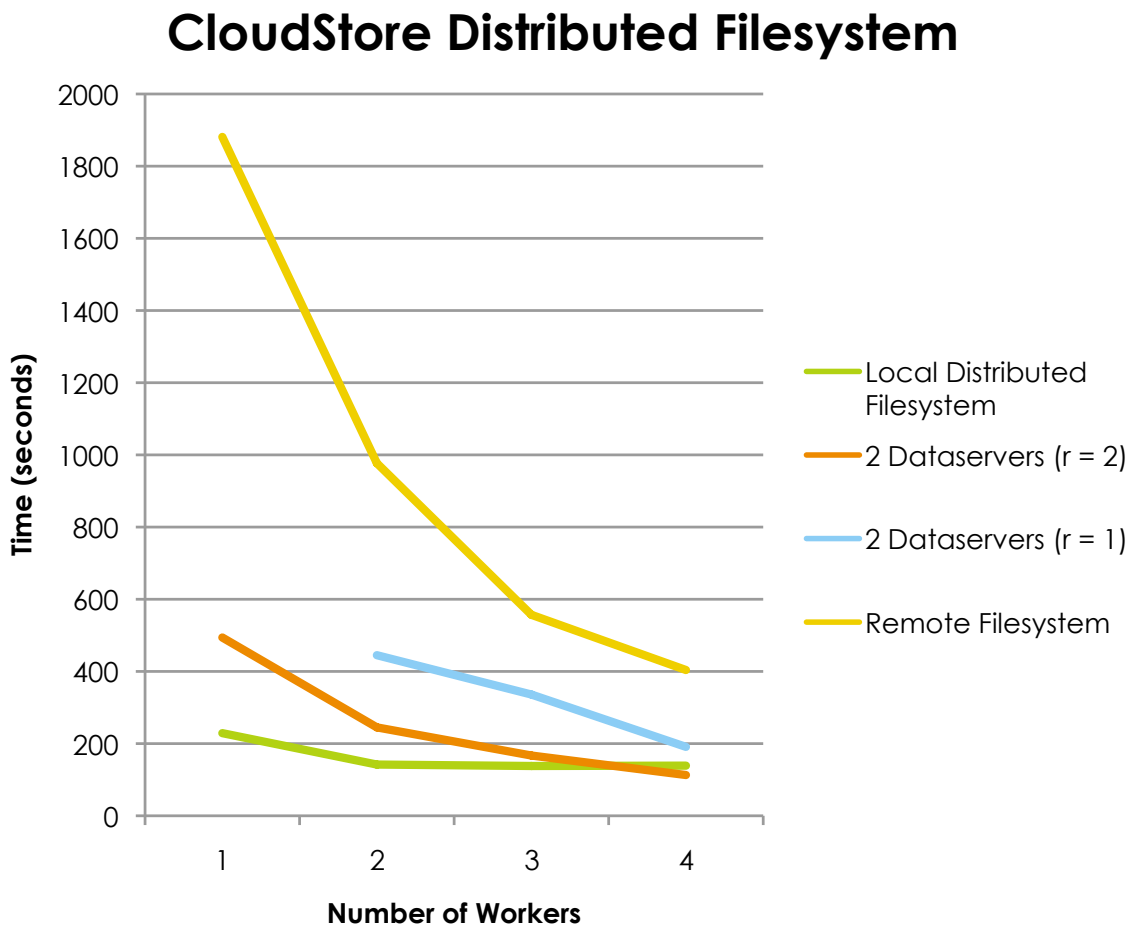
Objective: Intelligent Compute-Data Placement

- Objective: Intelligence in Compute-Data placement
- Strategies:
  - Assignment of workers (statically) determined by lowest $T_{transfer}$
    - Simple network measures (ping, throughput etc.) for $T_{transfer}$
  - Data pre-staging
    - Load-balancing (where easy to predict data requirement)
  - Assignment of workers , by minimizing data amount transferred
  - Placement of workers, based upon tracked dependencies

For a given objective, which strategy? Each strategy could have different mechanism used to implement…

Results: DFS (CloudStore)

- Read 2.3 gigabytes
- No coordination
- Handled multiple requests
- Some overhead for multiple dataservers in filesystem



SAGA-based frameworks: Logical ordering

Distributed Data Intensive Applications
e.g. Particle Physics, Astronomy, Bio-Informatics

AF

Programming Abstractions/Patterns/Higher-level APIs
e.g. MapReduce, All-Pairs, Scatter-Gather

RF

Common Runtime Support
e.g. Affinity, Fault Tolerance, Patterns

Physical Infrastructure
e.g. TeraGrid/XD, Clouds, Future Data Systems

SAGA-MapReduce

(GSOC'08 Miceli, Jha et al CCGrid'09; Merzky, Jha GPC'09)

- Interoperability: Use multiple infrastructure concurrently
- Control the N_W placement
  - Dynamic resource Allocation: "Map" phase different from "Reduce"
  - Distribution of data

T_s: Time-to-solution, including data-staging for SAGA-MapReduce (simple file-based mechanism)

Number-of-Workers			Size	T_s
TG	AWS	Eucalyptus	(MB)	(sec)
-	1	1	10	1.5
-	2	2	10	1.9
-	1	1	100	2.9
-	2	2	100	3.0
1	-	1	10	1.4
1	-	1	100	3.0
2	**2**	-	10	1.5
3	3	-	10	1.6
4	4	-	10	2.1
5	5	-	10	3.8

SAGA-MapReduce

(GSOC'08 Miceli, Jha et al CCGrid'09; Merzky, Jha GPC'09)

Controlling Relative Compute-Data Placement

Configuration compute	Configuration data	data size (GB)	work-load/worker (GB/W)	T_c (sec)
local	local-FS	1	0.1	466
distributed	local-FS	1	0.1	320
distributed	DFS	1	0.1	273.55
local	local-FS	2	0.25	673
distributed	local-FS	2	0.25	493
distributed	DFS	2	0.25	466
local	local-FS	4	0.5	1083
distributed	local-FS	4	0.5	912
distributed	DFS	4	0.5	848

TABLE I: Table showing T_c for different configurations of compute and data. The two compute configurations correspond to the situation where all workers are either placed locally or workers are distributed across two different resources. The data configurations arise when using a single local FS or a distributed FS (KFS) with 2 data-servers. It is evident from performance figures that an optimal value arises when distributing both data and compute.

Enhanced SAGA-MapReduce

(Erdelyim, Sehgal, Merzky, Jha GSoC'09 Project)

- Ease-of-use:
  - Simple, clean C++ API
  - Application only needs to link to a library
  - Custom input/output format support
    - Text-based and sequence file formats are already built-in
- Feature highlights:
  - Infrastructure-independent support for the MapReduce programming model
  - Native C++ library, no need to use wrappers like in case of Hadoop
  - Chaining of MapReduce jobs

Enhanced SAGA-MapReduce

(GSOC'08 Miceli, Jha et al CCGrid'09; Merzky, Jha GPC'09)

- Implementation details:
  - Efficient serialization support via Google's Protocol Buffers library
    - Trivial to extend Hadoop's librecordio or Thrift
  - Data-locality optimization with the help of SAGA's Replica API (soon)
  - Master-worker communication done through SAGA's Advert and Streams API

Wordcount (local advertdb)	8 workers/8 reducers(partitions)	*Old SAGA-MR*	Chunking time	*Enhanced SAGA-MR*	Workload/worker
LocalFS	1GB	3m16s	39s	3m15s	128M
	2GB	7m43s	1m28s	6m12s	256M
	4GB	14m39s	3m3s	12m51s	512M
	8GB	45m54s	6m15s	27m31s	1G

Sphere PM: Stream Processing

- Sphere – Generalized MapReduce
  - Tied closely with Sector
- Kernel – A (UDF) function that processes a single independent segment (file) in the stream
- Kernels execute in parallel to process the entire stream encompassing the data set
- Sphere supports the stream processing paradigm, allowing applications to:
  - Define a "kernel" function in a dynamic library (DLL)
  - Upload a stream of data to the Sector file system for processing

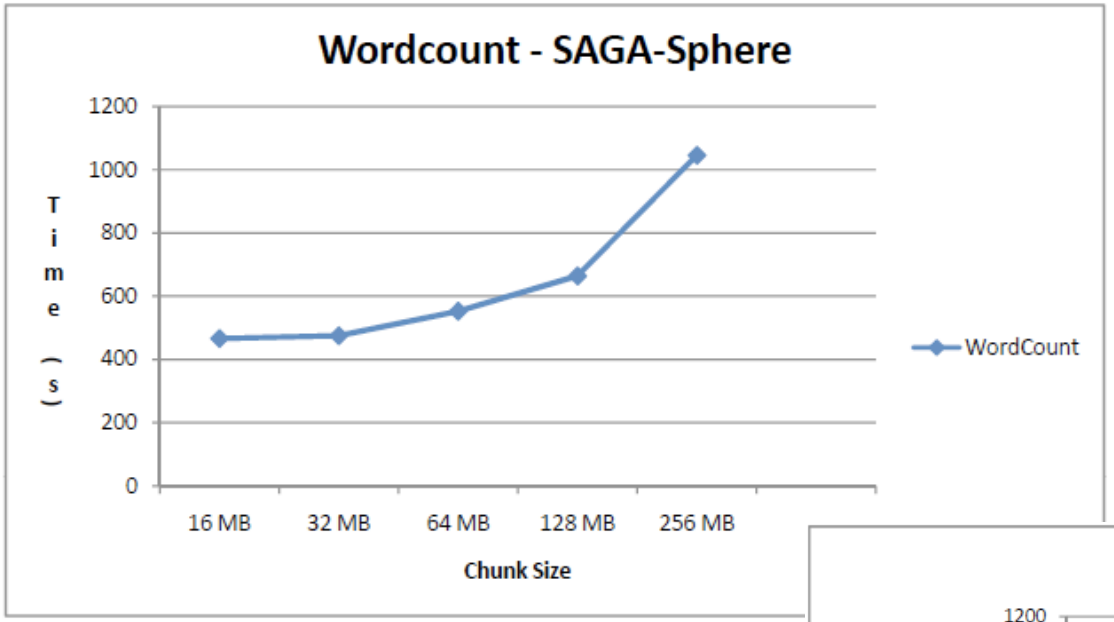
Integrating SAGA Sphere

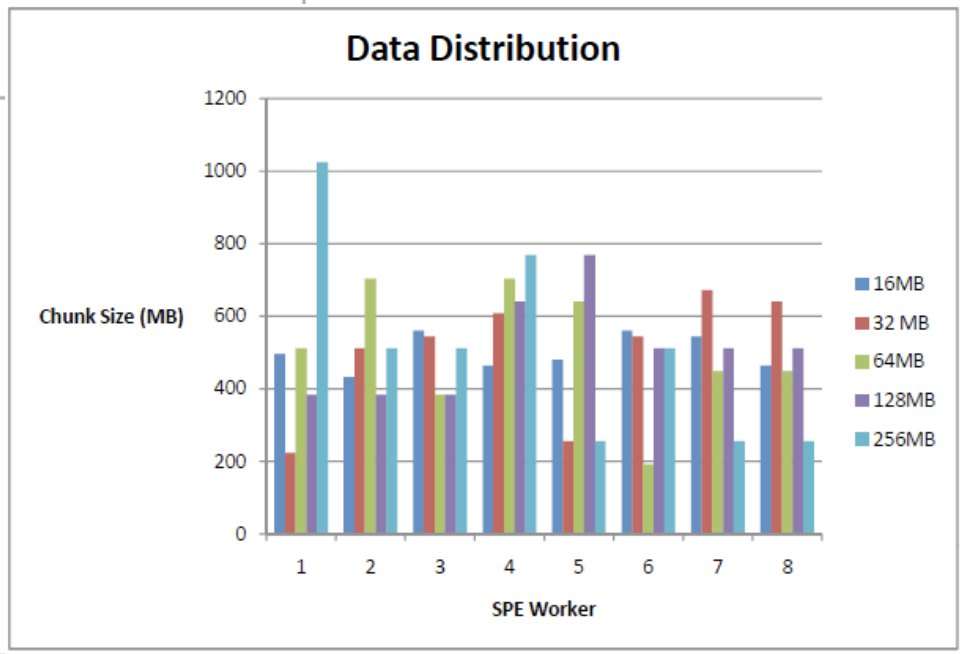
- Support for stream processing is provided through two adaptors: Sector and Sphere
- SAGA Sector file API can be used to:
  - Upload data sets to the Sector file system
  - Upload kernel functions wrapped in UDFs to Sphere
  - Download result sets to local disk after processing
- SAGA Sphere job API can be used to:
  - Execute kernel functions on uploaded data sets
  - Receive metrics on job completion times
- Authentication with the Sphere/Sector system is done implicitly.
  - Static and Dynamic authentication parameters are supported

Sector

- Sector – Data storage cloud designed for high speed networks
- Currently running on OCC test-bed that utilizes 10g/s WANs
- SAGA Sector adaptor translates SAGA File Package APIs into Sector operations
- Allows SAGA based applications to utilize the high performance OCC test-bed for data-intensive computations in conjunction with other DFS's like KFS, HDFS, GFS etc.

SAGA-Sphere WC
Sector FS





Dynamic Execution of DAGs

- Problem: Given a set of resources, where to schedule a set of independent/dependent DAG nodes?
- Goal: Decrease the makespan of the DAG i.e. Execution time
- To increase performance – data and compute placement must be taken into consideration
- Current scheduling heuristic based on:
  - Priority Assignment
  - Dynamic resource selection

Digedag: SAGA Workflow Package

- Digedag - prototype implementation of an experimental SAGA-based workflow package, with:
  - An API for programmatically expressing workflows
  - A parser for (abstract or concrete) workflow descriptions
  - An (in-time workflow) planner
  - A workflow enactor (using the SAGA engine)
- Use of an integrated API that allows the specification of the node and data dependencies to be specified & removes the need to manual (explicitly) build DAGs
- Can accept mDAG output, or Pegasus output
  - C-DAG output of digedag is general purpose, and can be used most simply without favour to say DAGman
- Move back and forth between A & C-DAG;

SAGA-based DAG Execution

Preserving Performance

#	resources	middleware	walltime	std-dev.	diff to local
1	l	f	68.7 s	9.4 s	– –
2	l	s	131.3 s	8.7 s	62.6 s
3	l	c	155.0 s	16.6 s	86.3 s
4	l	f, s	89.8 s	5.7 s	21.1 s
5	l	f, c	117.7 s	17.7 s	49.0 s
6	l	s, c	133.5 s	32.5 s	64.8 s
7	l	f, s, c	144.8 s	18.3 s	76.1 s
8	q	s	491.6 s	50.6 s	422.9 s
9	e	a	354.2 s	23.3 s	285.5 s
10	e, q	s, a	363.6 s	60.9 s	294.0 s
11	l, q, e	f, s, a	409.6 s	60.9 s	340.9 s
12	l	d	168.8 s[6]	5.3 s	100.1 s
11	p	d	309.7 s	41.5 s	241.0 s

TABLE II: Execution measurements
resources: *l=local,* ***p****=Purdue,* ***q****=Queen Bee,* ***e****=aws/EC2*
middleware: *f=fork/SAGA,* ***s****=ssh/SAGA,* ***a****=aws/SAGA,*
c=Condor/SAGA, ***d****=Condor/DAGMan,* ***p****=Pegasus*

Dynamic Execution of DAG

- Communication Overhead:
  - Use **netperf** UNIX tool to estimate data transfer rates between a set of resources.
  - Use this data to assign weight to DAG edges
- Priority assignments:
  - Higher priority assigned to nodes that contribute most to the communication overhead
  - Example: Parent node of the DAG CP has the highest priority
- Dynamic resource Selection:
  - **Min { Cost (n_x, r_y) = tlevel(n_x) + blevel(n_x) }**

n_x, node under consideration

r_y, resource from a set of given resources

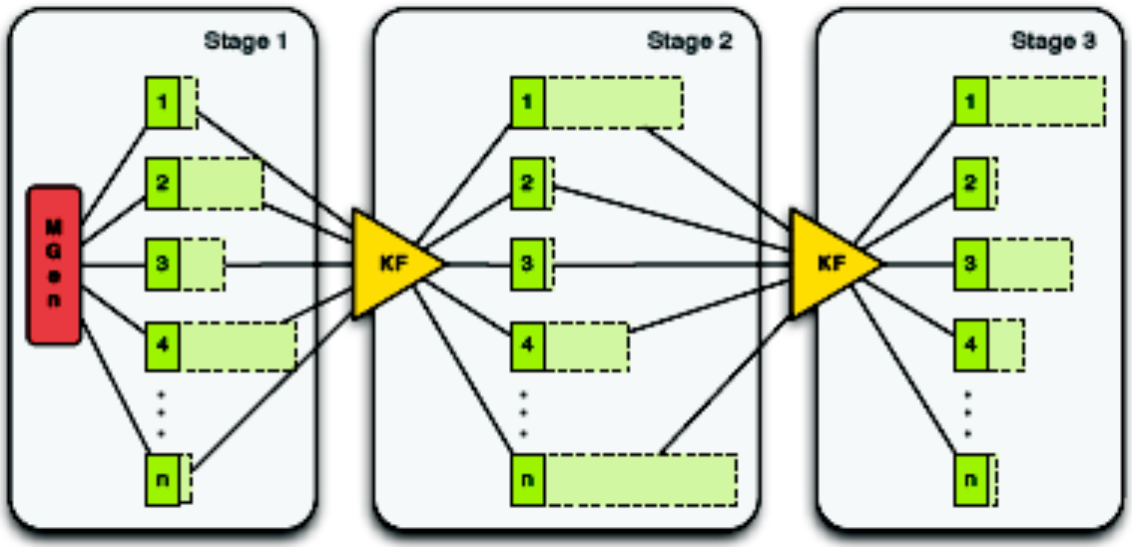
tlevel (n_x) , the heaviest top-level path for node n_x placed on r_y

blevel(n_x), the heaviest bottom-level path for node n_x placed on r_y

Ensemble Kalman Filters Heterogeneous Sub-Tasks

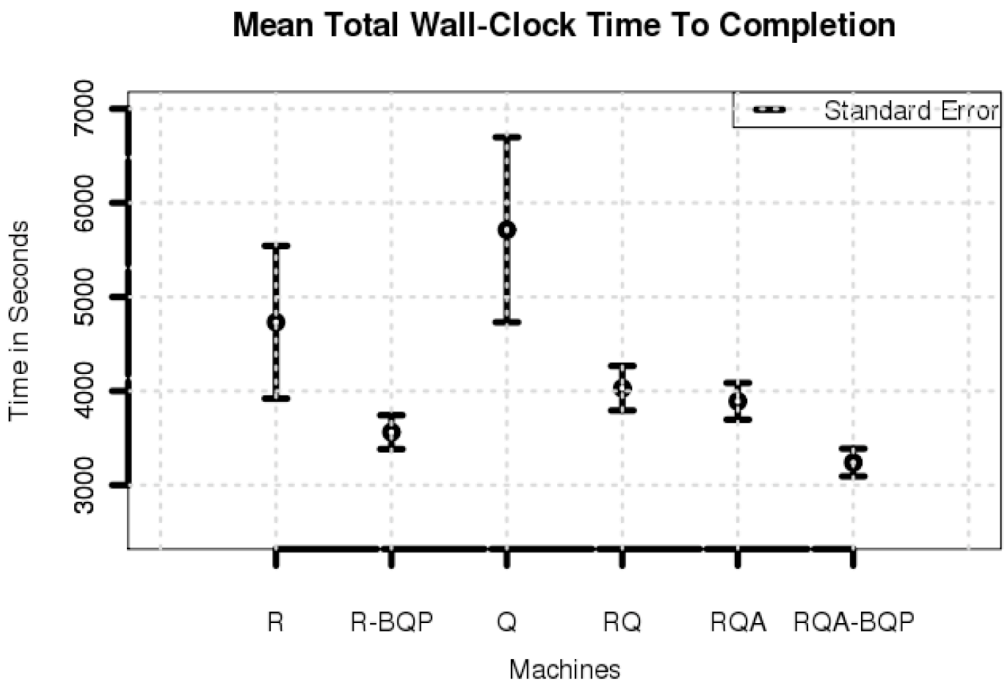
- Ensemble Kalman filters (EnKF), are recursive filters to handle large, noisy data; use the EnKF for history matching and reservoir characterization (Alan Sill gave a nice overview in the last MAGIC talk)

- EnKF is a particularly interesting case of irregular, hard-to-predict run time characteristics:



Results: Scale-Out Performance

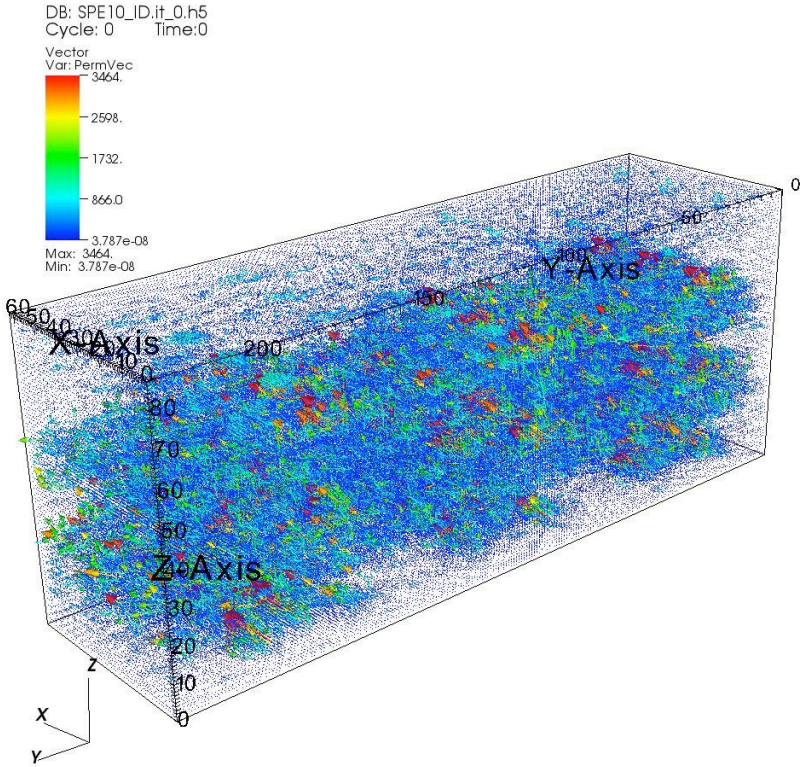
- Using more machines decreases the TTC and variation between experiments
- Using BQP decreases the TTC & variation between experiments further
- Lowest time to completion achieved when using BQP and all available resources



Khamra & Jha, GMAC, ICAC'09

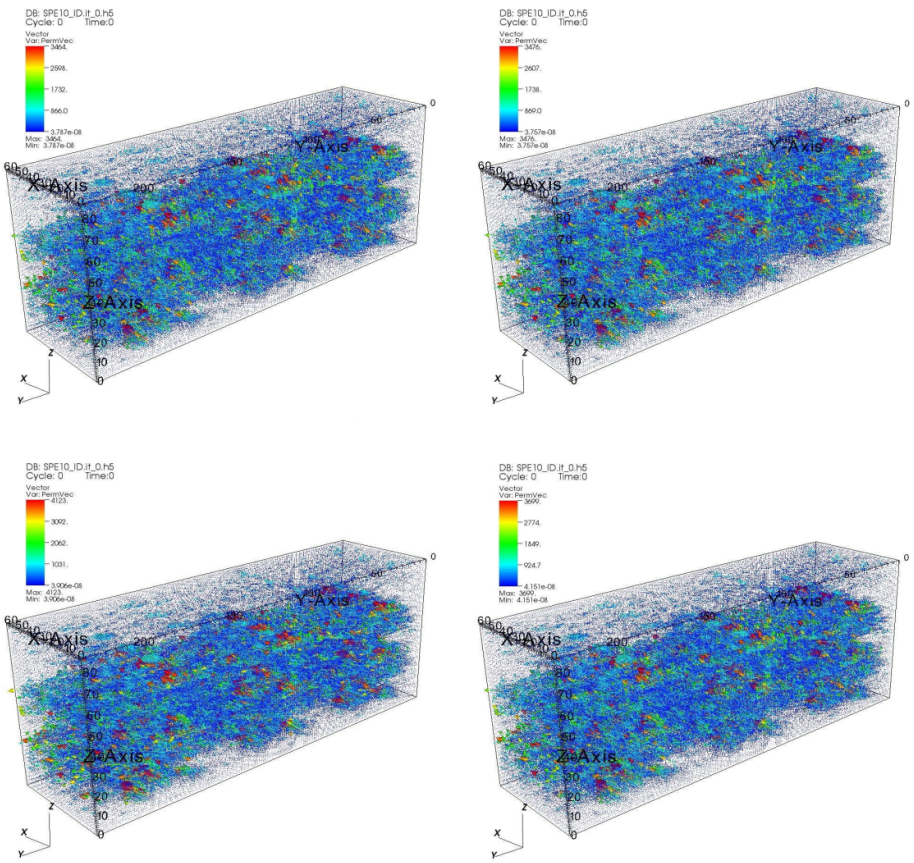
Extreme Distribution: Frameworks?

- History match on a 1 million grid cell problem, with a thousands of ensemble members
- The entire system will have a few billion degrees of freedom
- This will increase the need for scale-out, autonomy, fault tolerance, self healing etc…



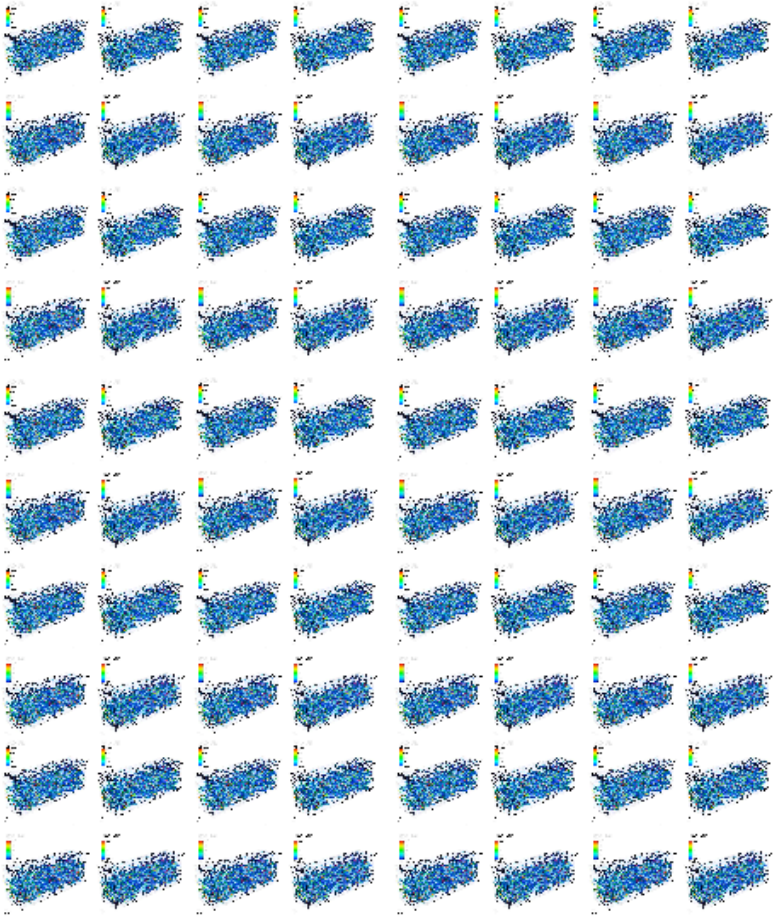
Extreme Distribution: Frameworks?

- History match on a 1 million grid cell problem, with a thousand ensemble members
- The entire system will have a few billion degrees of freedom
- This will increase the need for scale-out, autonomy, fault tolerance, self healing etc…



Extreme Distribution: Frameworks?

- History match on a 1 million grid cell problem, with a thousand ensemble members
- The entire system will have a few billion degrees of freedom
- This will increase the need for scale-out, autonomy, fault-tolerance, self healing etc…



LSU
CENTER FOR COMPUTATION & TECHNOLOGY

Does SAGA Provide A Fresh Perspective?

Application	*Interoperabilty*	*Scale-Out*	*Extensibility*	*Adaptivity*	*Simplicity*
SAGA-Montage	HPC-Condor-Clouds	Yes	FAUST	-	Yes
Replica-Exchange	Multiple TG nodes	Yes	Yes	Yes	Yes
Multi-Physics	Multiple HPC, Condor	Yes	Load-Balancing	Yes	Yes
Reservoir Simulation (EnKF)	Multiple TG-Condor-Cloud	Yes	Yes (BQP)	Yes	Yes
SAGA-MapReduce	TG-Clouds (ECP+EC2)	Yes	Yes	Yes (C2D)	Lower Performance

Table 3: **Fresh Perspective on Distributed Applications and CyberInfrastructure**

Interoperability *Ability to work across multiple distributed resources*
Scale-Out *Ability to use multiple distributed resources concurrently*
Extensibility *Support new patterns/abstractions, different programming systems and Infrastructure*
Adaptivity *Response to fluctuations in dynamic resources and availabilty of data*
Simplicity *Ease of above, at different levels and without sacrificing functionality or performance*

Conclusions

- Discussed SAGA-based approaches to developing and executing distributed DIA
- Range of infrastructure & dynamic behaviour is large – intrinsic (application) and extrinsic (system/resource)
  - Responding can have important performance consequences
- Proposed a logical ordering for designing/developing Frameworks
  - Provides the basis with which to perform experiments to understand performance trade-offs, configurations, infrastructure for a broad range of applications
  - Designing and Implementing such frameworks that support IDEAS is a challenging endeavour

LSU
CENTER FOR COMPUTATION & TECHNOLOGY

Acknowledgements

SAGA Team and DPA Team and the UK-EPSRC (UK EPSRC: DPA, OMII-UK , OMII-UK PAL), NSF (HPCOPS, Cybertools) and LA-BOR

People:

SAGA D&D: Hartmut Kaiser, Ole Weidner, Andre Merzky, Joohyun Kim, Lukasz Lacinski, JoãoAbecasis, *Chris Miceli, Bety Rodriguez-Milla*

SAGA Users: Andre Luckow, Yaakoub el-Khamra, Kate Stamou, Cybertools (AbhinavThota, Jeff, N. Kim), OwainKenway

Google SoC: *Michael Miceli, Saurabh Sehgal, MiklosErdelyi*

Collaborators and Contributors: Steve Fisher & Group, Sylvain Renaud (JSAGA), Go Iwai & Yoshiyuki Watase (KEK)

DPA: **Dan Katz**, Murray Cole, Manish Parashar, Omer Rana, **Jon Weissman**

DIC: **Dan Katz and Jon Weissman**