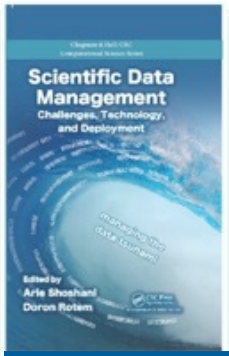
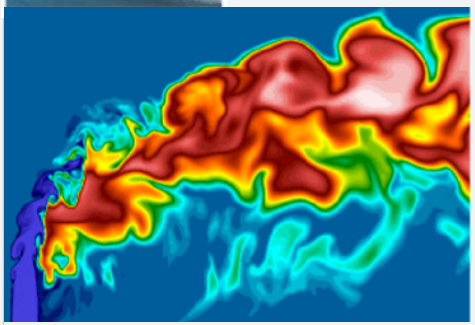
ADIOS: An I/O framework for exascale platforms





Ultrascale Visualization Workshop

11/13/2011

Work supported under:

ASCR : CPES, SDM, Runtime Staging, SAP, OLCF, Co-design

OFES : GPSC, GSEP

NSF : EAGER, RDAV

NASA : ROSES

Scott A. Klasky

klasky@ornl.gov

H. Abbasi[1], S. Ethier[8], R. Grout[7], Q. Liu[2], J. Logan[1], J. Lofstead[5], K. Moreland[5], M. Parashar[6], N. Podhorszki[1], N. Samatova[10], K. Schwan[4], A. Shoshani[3], R. Vatsavai[1], M. Wolf[4], J. Wu[3], W. Yu[9]

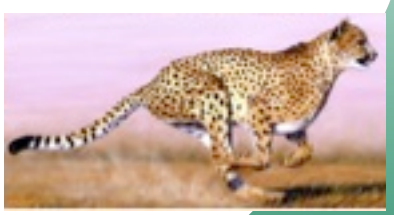
[1]ORNL, [2] U.T. Knoxville, [3]LBNL, [4]Georgia Tech, [5]Sandia Labs, [6] Rutgers, [7]NREL, [8]PPPL, [9]Auburn University, [10]NCSU

Outline

- Requirements for an I/O system
  - Why not just make a better file system?
  - Why not just make a better visualization system?
  - Why not just make a better scientific application?
- ADIOS 101
  - What is ADIOS
  - ADIOS performance
  - ADIOS examples
- Building a next generation I/O system
  - Service Oriented Architecture
  - Staging
  - Moving work to data
- Next Steps
  - Building an in-transit workflow engine

We want our system to be so easy, even a chimp can use it

Sustainable
Fast
Scalable
Portable



Extreme scale computing

- Trends
  - More FLOPS
  - Limited number of users at the extreme scale
- Problems
  - Performance
  - Resiliency
  - Debugging
  - Getting Science done
- Problems will get worse
  - Need a "revolutionary" way to store, access, debug to get the science done!
- ASCI purple (49 TB/140 GB/s) – JaguarPF (300 TB/ 200 GB/s)

Systems	20			
System Peak Flops/s				
System Memory				
Node Performance				
Node Memory BW				/s
Node Concurrency				(1000)
Interconnect BW	1.5 GB/			50 GB/s
System Size (Nodes)	18,700	100,0	,000	O(Million)
Total Concurrency	225,000	3 Millio	50 Million	O(Billion)
Storage	15 PB	30 PB	150 PB	300 PB
I/O	0.2 TB/s	2 TB/s	10 TB/s	20 TB/s
MTTI	Days	Days	Days	O(1Day)
Power	6 MW	~10 MW	~10 MW	~20 MW

Most people get < 5 GB/s at scale

ILLUSTRATION: A. TOVEY

From J. Dongarra, "Impact of Architecture and Technology for Extreme Scale on Software and Algorithm Design," Cross-cutting Technologies for Computing at the Exascale, February 2-5, 2010.

Next generation I/O and file system challenges

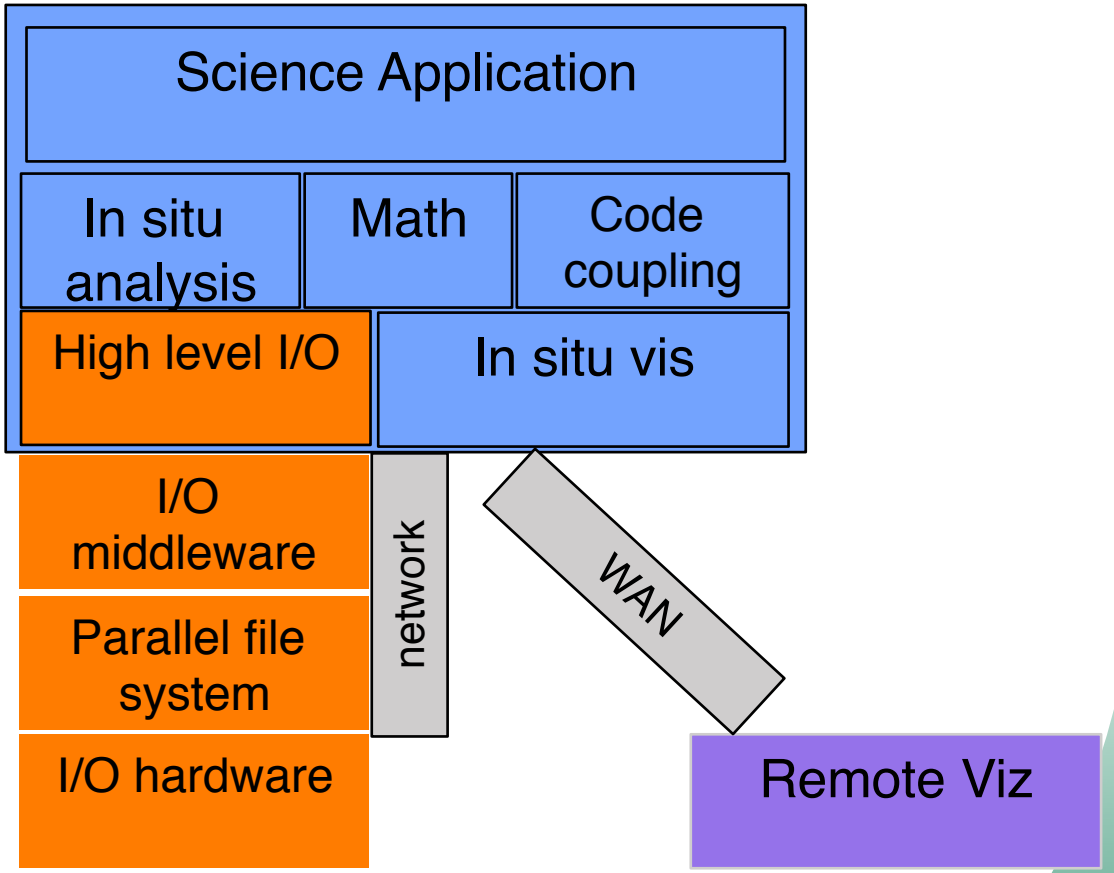
- At the architecture or node level
  - Use increasingly deep memory hierarchies coupled with new memory properties
- At the system level
  - Cope with I/O rates and volumes that stress the interconnect and can severely limit application performance
  - Can consume unsustainable levels of power
- At the exascale
  - Immense aggregate I/O needs with potentially uneven loads placed on underlying resource
  - Can result in data hotspots, interconnect congestion and similar issues

File Systems

File system	Hard links	Symbolic links	Block journaling	Metadata-only journaling	Case-sensitive	Case-preserving	File Change Log	Snapshot	XIP	Encryption	COW	integrated LVM	Data deduplication	Volumes are resizeable
CP/M file system	No	No	No	No	No	No	No	No	No	No	No	No	No	Unknown
DECtape	No	No	No	No	No	No	No	No	No	No	No	No	No	Unknown
Level-D	No	No	No	No	No	No	No	No	No	No	Unknown	Unknown	Unknown	Unknown
RT-11	No	No	No	No	No	No	No	No	No	No	No	No	No	Unknown
DOS (GEC)	No	No	No	No	No	No	No	No	No	No	No	No	No	Unknown
OS4000	No	Yes[71]	No	No	No	No	No	No	No	No	No	No	No	Unknown
V6FS	Yes	No	No	No	Yes	Yes	No	No	No	No	No	No	No	Unknown
V7FS	Yes	No[72]	No	No	Yes	Yes	No	No	No	No	No	No	No	Unknown
FAT12	No	No	No	No	No	Partial	No	No	No	No	No	No	No	Offline[73]
FAT16	No	No	No	No	No	Partial	No	No	No	No	No	No	No	Offline[73]
FAT32	No	No	No	No	No	Partial	No	No	No	No	No	No	No	Offline[73]
exFAT	No	No	Unknown	No	No	Yes	No	Unknown	Unknown	No	Unknown	Unknown	Unknown	Unknown
GFS	Yes	Yes[74]	Yes	Yes[75]	Yes	Yes	No	No	No	No	Unknown	Unknown	Unknown	Unknown
GPFS	Yes	Yes	Unknown	Unknown	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	No	Online
HPFS	No	No	No	No	No	Yes	No	Unknown	No	No	Unknown	Unknown	No	Unknown
NTFS	Yes	Yes[76]	No[77]	Yes[77]	Yes[78]	Yes	Yes	Partial[79]	Yes	Yes	Partial	Unknown	No	Online[80]
HFS	No	Yes[81]	No	No	No	Yes	No	No	No	No	No	No	No	Unknown
HFS Plus	Yes[82]	Yes	No	Yes[83]	Partial[84]	Yes	Yes[85]	No	No	No[86]	No	No	No	Offline
FFS	Yes	Yes	No	No[87]	Yes	Yes	No	No	No	No	No	No	No	Offline (cannot be shrunk)[88]
UFS1	Yes	Yes	No	No	Yes	Yes	No	No	No	No	No	No	No	Unknown
UFS2	Yes	Yes	No	No[89][90]	Yes	Yes	No	Yes	Unknown	No	No	No	No	Offline (cannot be shrunk)[91]
LFS	Yes	Yes	Yes[92]	No	Yes	Yes	No	Yes	No	No	Unknown	Unknown	Unknown	Unknown
ext2	Yes	Yes	No	No	Yes	Yes	No	No	Yes[93]	No	No	No	No	Online[94]
ext3	Yes	Yes	Yes[95]	Yes	Yes	Yes	No	No	Yes	No	No	No	No	Online[94]
ext4	Yes	Yes	Yes[95]	Yes	Yes	Yes	No	No	Yes	No	No	No	No	Online[94]
Lustre	Yes	Yes	Yes[95]	Yes	Yes	Yes	Yes in 2.0	No[58]	No	No[58]	No[58]	No[58]	No[58]	Online[96]
NILFS	Yes	Yes	Yes[92]	No	Yes	Yes	Yes	Yes	No	No	Yes	Unknown	Unknown	Unknown
ReiserFS	Yes	Yes	No[97]	Yes	Yes	Yes	No	No	No	No	No	No	No	Offline
Reiser4	Yes	Yes	Yes	No	Yes	Yes	No	Unknown	No	Yes[98]	Yes	No	Unknown	Online (can only be shrunk offline)
OCFS	No	Yes	No	No	Yes	Yes	No	No	No	No	Unknown	Unknown	Unknown	Unknown
OCFS2	Yes	Yes	Yes	Yes	Yes	Yes	No	Partial[99]	No	No	Unknown	No	No	Online for version 1.4 and higher
Reliance	No	No	No[100]	No	No	Yes	No	No	No	No	Yes	No	No	Unknown
Reliance Nitro	Yes	Yes	No[100]	No	Depends on OS	Yes	No	No	No	No	Yes	No	No	Unknown
XFS	Yes	Yes	Yes	Yes	Yes[101]	Yes	No	No	No	No	No	No	No	Online (cannot be shrunk)
JFS	Yes	Yes	No	Yes	Yes[102]	Yes	No	Yes	No	No	Yes	Unknown	Unknown	Online (cannot be shrunk)[103]
QFS	Yes	Yes	No	Yes	Yes	Yes	No	No	No	No	Unknown	Unknown	Unknown	Unknown
Be File System	Yes	Yes	No	Yes	Yes	Yes	Unknown	No	No	No	No	No	No	Unknown
NSS	Yes	Yes	Unknown	Yes	Yes[104]	Yes[104]	Yes[105]	Yes	No	Yes	Unknown	Unknown	Unknown	Unknown
NWFS	Yes[106]	Yes[106]	No	No	Yes[104]	Yes[104]	Yes[105]	Unknown	No	No	No	Yes[107]	Unknown	Unknown
ODS-2	Yes	Yes[108]	No	Yes	No	No	Yes	Yes	No	No	Unknown	Unknown	Unknown	Unknown
ODS-5	Yes	Yes[108]	No	Yes	No	Yes	Yes	Yes	Unknown	No	Unknown	Unknown	Unknown	Unknown
UDF	Yes	Yes	Yes[92]	Yes[92]	Yes	Yes	No	No	Yes	No	No	No	No	Unknown
VxFS	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes[109]	Unknown	No	Unknown	Unknown	Unknown	Unknown
Fossil	No	No	No	No	Yes	Yes	Yes	Yes	No	No	Unknown	No	Yes[110]	Unknown
ZFS	Yes	Yes	Yes[111]	No[111]	Yes	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Online (cannot be shrunk)[112]
VMFS2	Yes	Yes	No	Yes	Yes	Yes	No	No	No	No	Unknown	Unknown	Unknown	Unknown
VMFS3	Yes	Yes	No	Yes	Yes	Yes	No	No	No	No	Unknown	Unknown	Unknown	Unknown
Btrfs	Yes	Yes	Unknown	Yes	Yes	Yes	Unknown	Yes	No	Planned[113]	Yes	Yes	Work-in-Progress	Online
File system	Hard links	Symbolic links	Block journaling	Metadata-only journaling	Case-sensitive	Case-preserving	File Change Log	Snapshotting	XIP	Encryption	COW	integrated LVM	Data deduplication	Volumes are resizeable

Tools and Technologies to work with large data

- MPI
- ROMIO
- HDF5
- Netcdf-4
- Globus on-line
- SciDB
- Paraview
- Visit
- …



But ….

- We want to view I/O as not just “I/O” but rather
  - I/O pipelines
- They need semantic knowledge
  - It’s just not a bunch of bytes
  - How do you interpret the information to analyze it? To visualize it?
- This tells us that we need self-describing files

But ….

- Data is getting large, and produced from many sources (MPI procs, sensors, …)
- Data can be analyzed, and visualized
  - In-situ
  - In-transit
  - Co-processing
  - On-clusters
  - Clouds
  - Desktops
  - Smart-phones

So…

- We need self-describing data streams(chunks) from many **process**es which can
  - Have embedded code
  - Have semantic knowledge of how to work with it
  - Have embedded workflows
  - Embed provenance information

And

- Data naturally comes in **group**s
  - Check-point restart data
  - Analysis data
  - Visualization data
  - Monitoring data

But how do we make this easy-to-use

- We need to create schema's (standards) that doesn't involve much user involvement
  - For in-transit visualization, analysis
  - In-situ visualization, analysis
  - Co-processing visualization, analysis
  - For code-coupling
  - For any operations on the data which do NOT require human intervention

Development

- We want to have systems people build the systems layer
  - Viz people build the viz. layer
  - Analysis people build the analysis layer
  - …
- Ideally we need to create "teams" to create the software
  - Teams for the apps
  - Teams for the analysis
  - Teams for the math
  - Teams for data management
  - Teams for visualization

But…

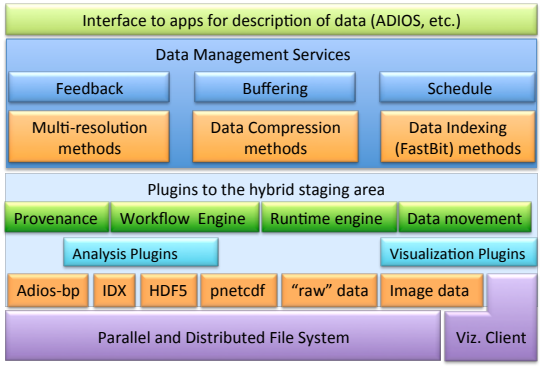
- All of these teams have a diverse set of developers
- All of these components need to be created independently
- Debugged and Tested independently
- All of these components need to be stand-alone, and easily integrated in the complex system
  - On 1 machine
  - Integrated on many machines
- And of of these components need to work together in one (or many small) workflows

Thus: the SOA philosophy

- The overarching design philosophy of our framework is based on the Service-Oriented Architecture
  - Used to deal with system/application complexity, rapidly changing requirements, evolving target platforms, and diverse teams
- Applications constructed by assembling services based on a universal view of their functionality using a well-defined API
- Service implementations can be changed easily
- Integrated simulation can be assembled using these services
- Manage complexity while maintaining performance/scalability
  - Complexity from the problem (complex physics)
  - Complexity from the codes and how they are
  - Complexity of underlying disruptive infrastructure
  - Complexity from coordination across codes and research teams

Adaptable I/O System

- Provides portable, fast, scalable, easy-to-use, metadata rich output with a simple API
- Change I/O method by changing XML input file
- Layered software architecture:
  - Allows plug-ins for different I/O implementations
  - Abstracts the API from the method used for I/O
- Open source: 3rd release: first beta in 2007
  - http://www.nccs.gov/user-support/center-projects/adios/
- High Writing Performance
  - S3D: 32 GB/s with 96K cores, 1.9MB/core: 0.6% I/O overhead with ADIOS
  - XGC1 code → 40 GB/s, SCEC code 30 GB/s
  - GTC code → 40 GB/s, GTS code: 35 GB/s



ADIOS API

Code which writes data

```
call adios_open (adios_handle,
    "writer2D", filename, "w",
    group_comm, adios_err)
#include "gwrite_writer2D.fh"
call adios_close (adios_handle,
    adios_err)
```

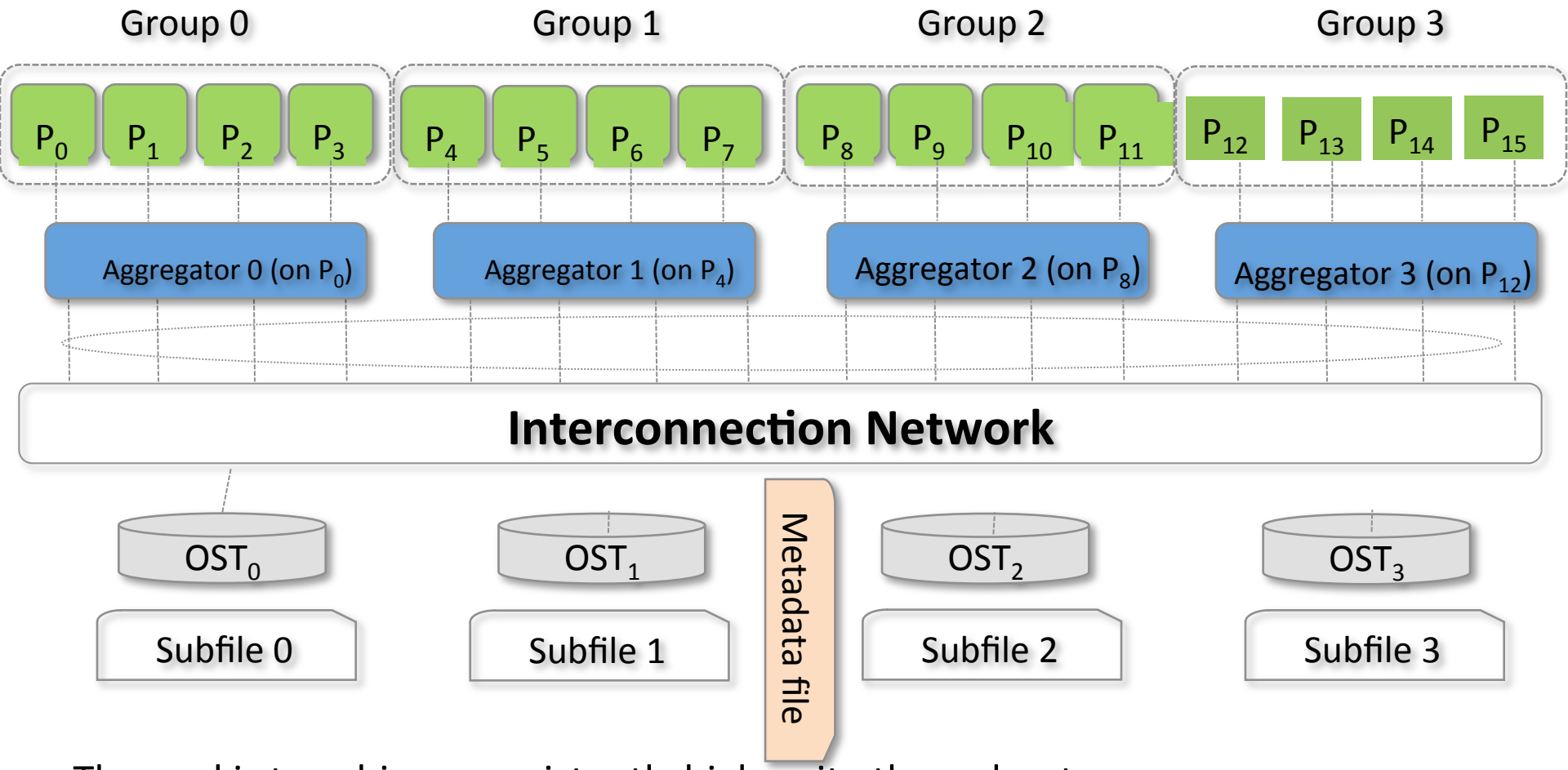
XML for mapping C/F90 variables

```
<adios-group name="writer2D" >
 <global-bounds dimensions="gdx,gdy"
      offsets="ox,oy">
   <var name="xy" type="real"
        dimensions="ldx,ldy"/>
</global-bounds>
</adios-group>
<transport group="writer2D"
       method = "MPI" />
```

Code which reads data

```
call adios_set_read_method (BP    ,ierr)
call adios_read_init (group_comm, ierr)
call adios_fopen (fh, fn, group_comm, gcnt, adios_err)
call adios_gopen (fh, gh, "writer2D", vcnt, acnt, adios_err)
call adios_read_var (gh, "gdx", offset, readsize, gdx, read_bytes)
call adios_read_var (gh, "gdy", offset, readsize, gdy, read_bytes)
! … calculate offsets and sizes of xy to read in…
call adios_read_var (gh, "xy", offset, readsize, xy, read_bytes)
call adios_gclose (gh, adios_err)
call adios_fclose (fh, adios_err)
```

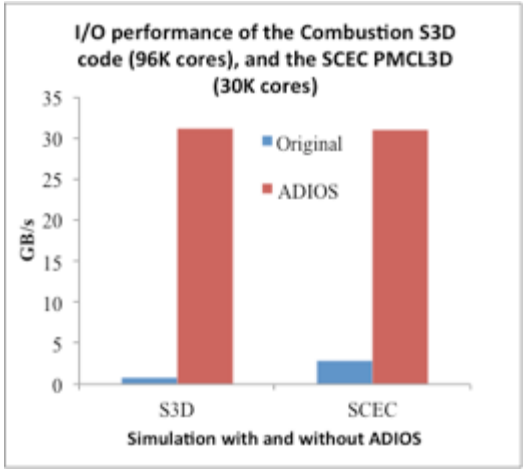
ADIOS write method

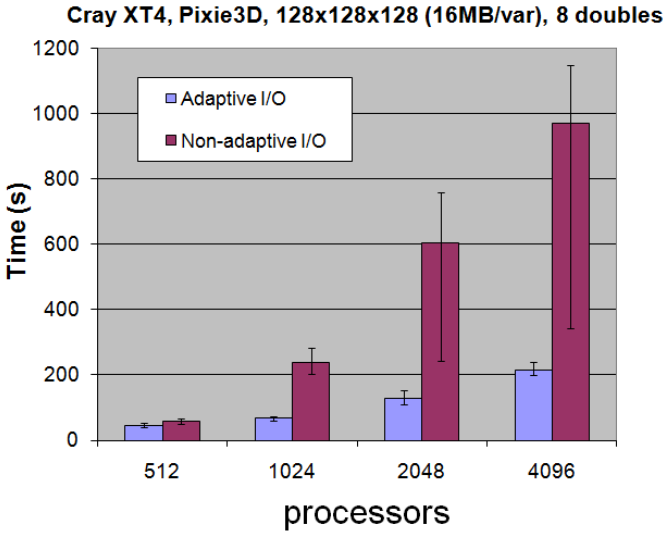


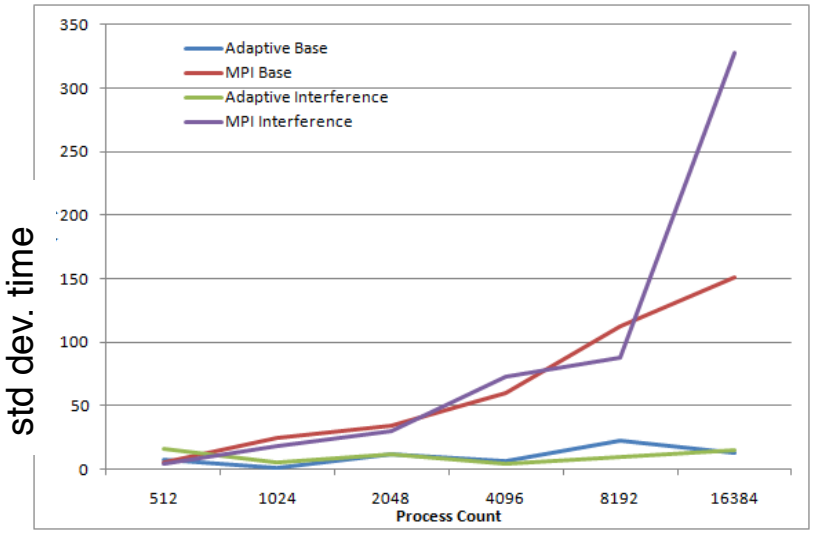
- The goal is to achieve consistently high write throughput
- Aggregating data using brigade communication
- The adaptive control is achieved by controlling to which downstream node the data is sent. If one group has finished writing to disk, the unfinished data from other groups can be shifted to it.

Performance Optimizations for Write

- New technologies are usually constrained by the lack of usability in extracting performance
- Next generation I/O frameworks must address this concern
  - Partitioning the task of optimizations from the actual description of the I/O
- Innovate to deal with high I/O variability, and increase the "average" I/O performance

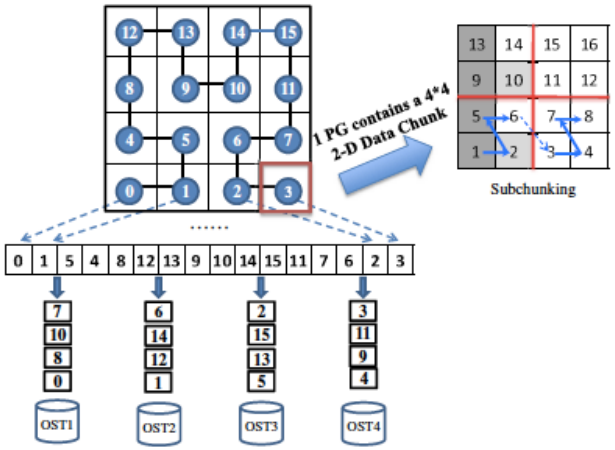






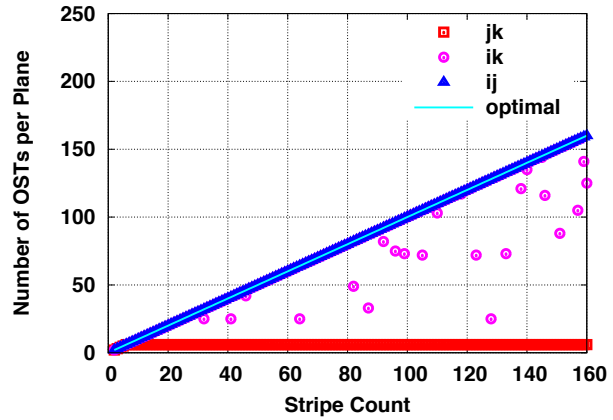
Understand why: (Examine reading a 2D plane from a 3D dataset)

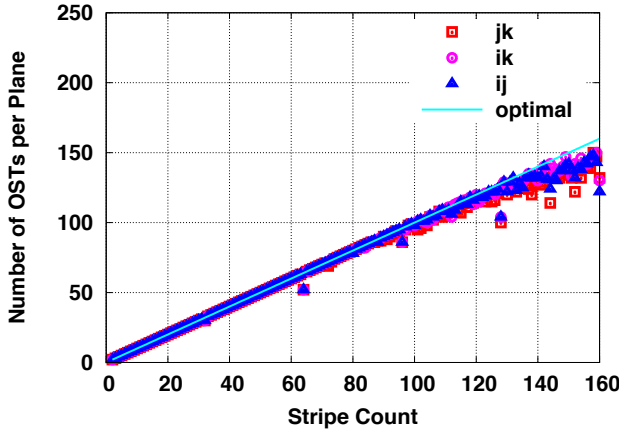
- Use Hilbert curve to place chunks on lustre file system with an Elastic Data Organization



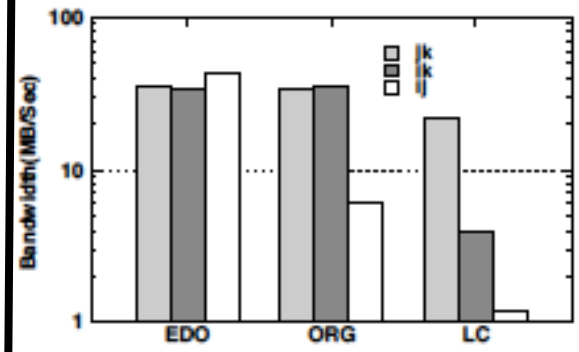
(b) SFC-based Placement with EDO

Theoretical concurrency

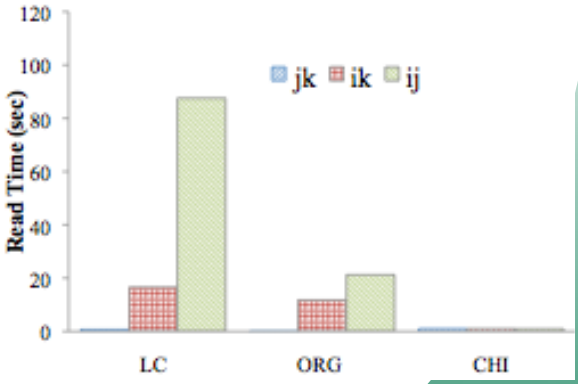




Observed Performance

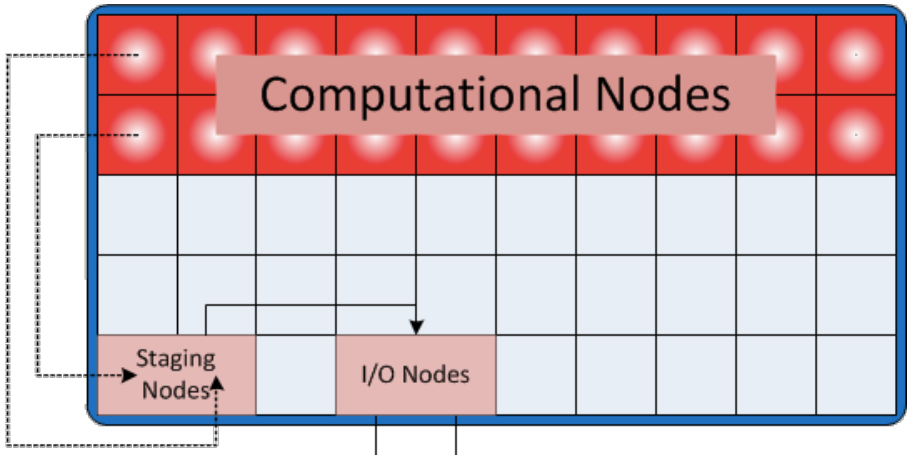


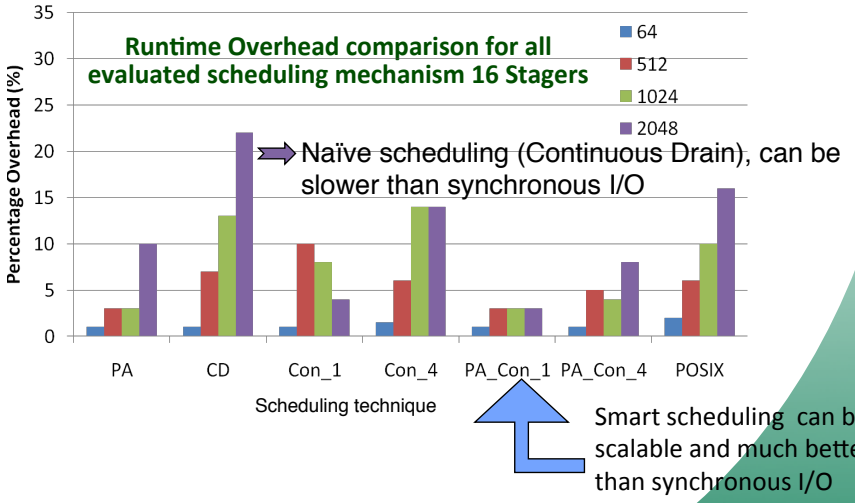
(a) Case S (stripe=128)



Designing the system

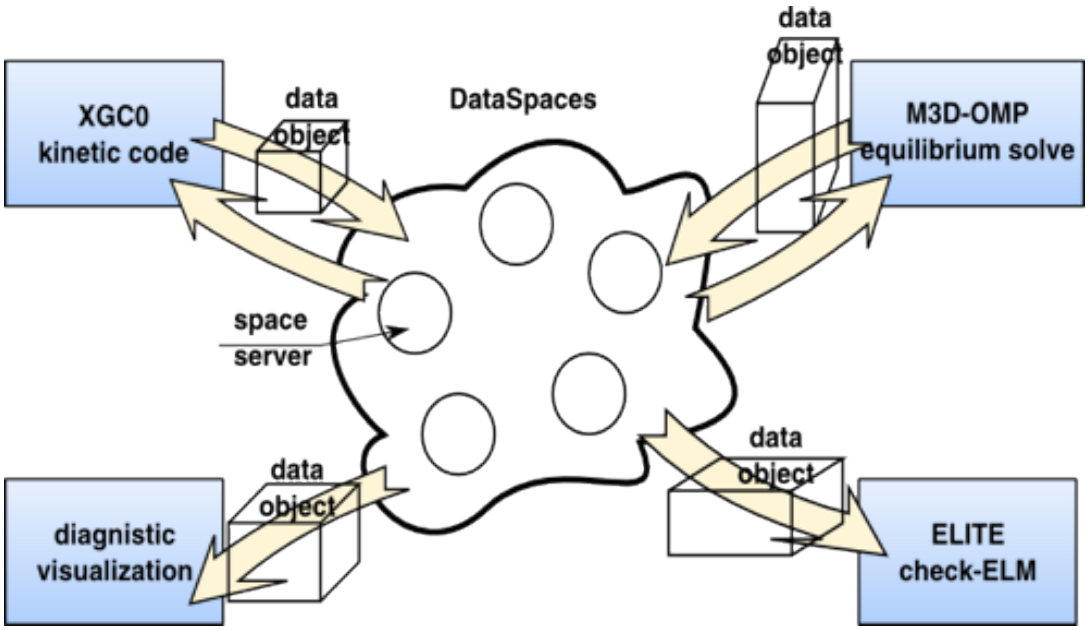
- Why staging?
  - Decouple file system performance variations and limitations from application run time
- Enables optimizations based on dynamic number of writers
- High bandwidth data extraction from application: RDMA
- Scalable data movement with shared resources requires us to manage the transfers





Approach allows for in-memory code coupling

- Semantically-specialized virtual shared space
- Constructed on-the-fly on the cloud of staging nodes
  - Indexes data for quick access and retrieval
- Complements existing interaction/coordination mechanisms
- In-memory code coupling becomes part of the I/O pipeline



Change to staging:

Code which writes data

```
call adios_open (adios_handle,
    "writer2D", filename, "w",
    group_comm, adios_err)
#include "gwrite_writer2D.fh"
call adios_close (adios_handle,
    adios_err)
```

XML for mapping C/F90 variables

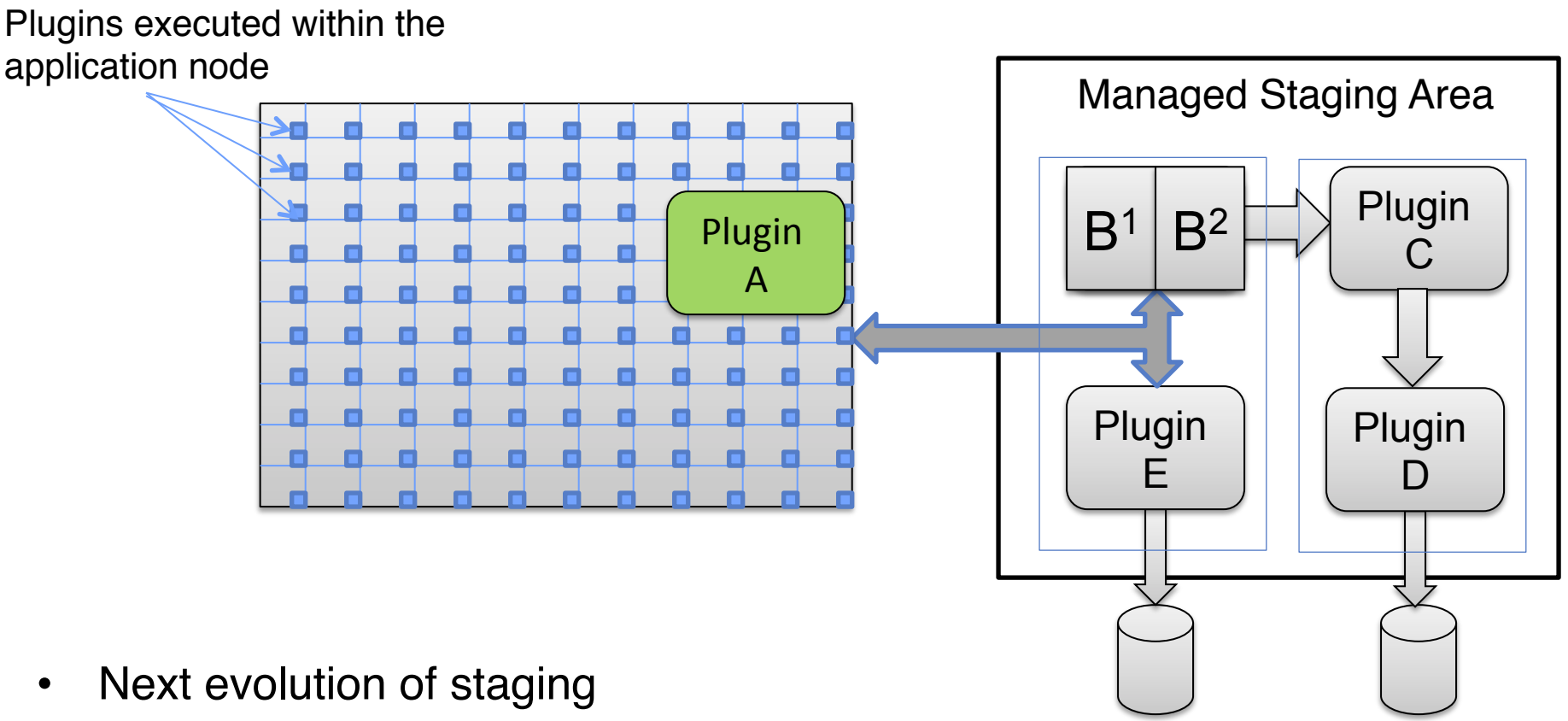
```
<adios-group name="writer2D" >
 <global-bounds dimensions="gdx,gdy"
      offsets="ox,oy">
   <var name="xy" type="real"
        dimensions="ldx,ldy"/>
</global-bounds>
</adios-group>
<transport group="writer2D"
       method = "DART"/>
```

Code which reads data

```
call adios_set_read_method (DART ,ierr)
call adios_read_init (group_comm, ierr)
call adios_fopen (fh, fn, group_comm, gcnt, adios_err)
call adios_gopen (fh, gh, "writer2D", vcnt, acnt, adios_err)
call adios_read_var (gh, "gdx", offset, readsize, gdx, read_bytes)
call adios_read_var (gh, "gdy", offset, readsize, gdy, read_bytes)
! … calculate offsets and sizes of xy to read in…
call adios_read_var (gh, "xy", offset, readsize, xy, read_bytes)
call adios_gclose (gh, adios_err)
call adios_fclose (fh, adios_err)
```

Now we have memory to memory coupling

Hybrid staging



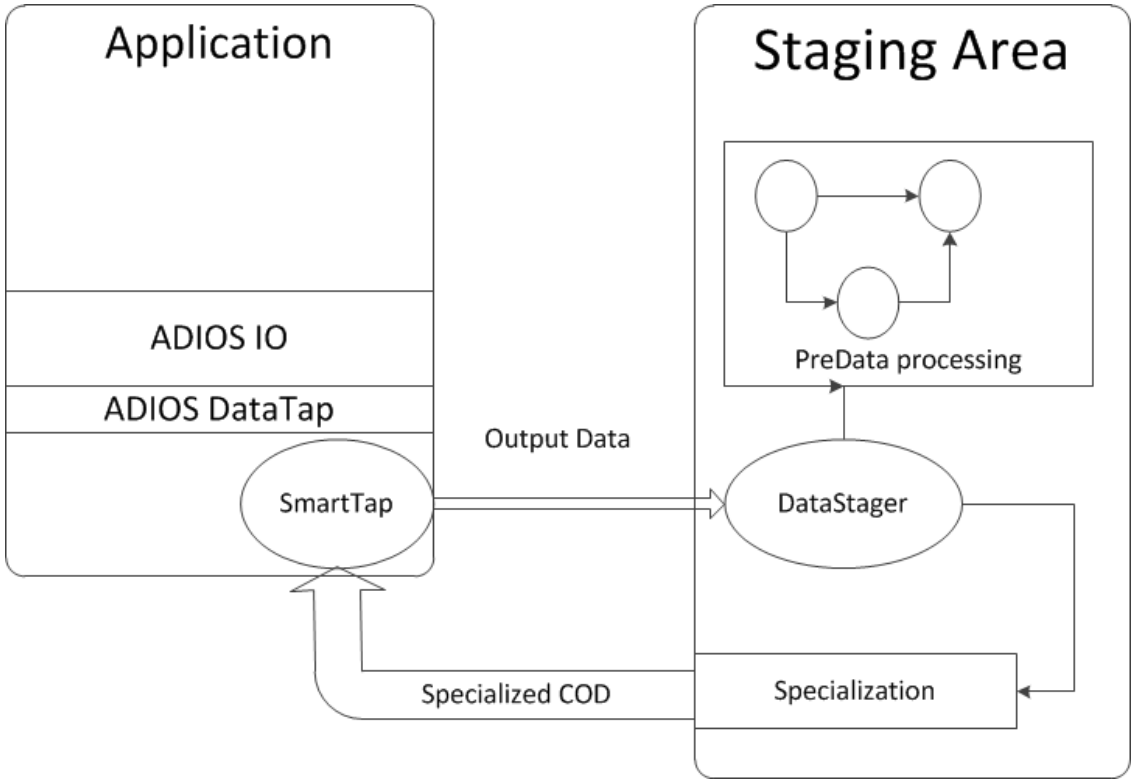
- Next evolution of staging
- Move plugins/services back to application
- Largest computational capacity
- Integrate with plugins in the remote staging area
- Need runtime system to schedule service, and need a programming model that simplifies the building and execution of the services

But we need to still reduce the amount of data

- We can use both lossy and lossless compression methods to reudce the data
  - ISABELA, ISOBAR, ….
- Can reduce the amount of data by 10X (lossy) for some datasets, with set accuracy
- But still need to index to query: Reduce the amount data being sent over the network
  - FastBit, ISABELA-QA, …
- We can also generate a multi-resolution stream-based compression method
  - Good way to allow users to choose ROI and zoom into higher resolution versions when necessary

But we can move work to data (I/II)

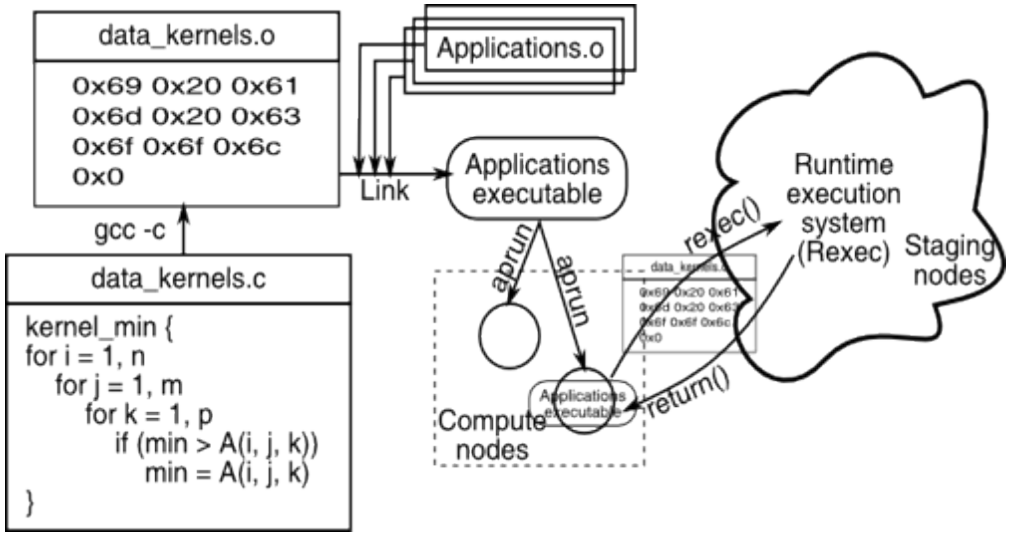
- Runtime placement decisions
- Dynamic code generation
- Filter specialization

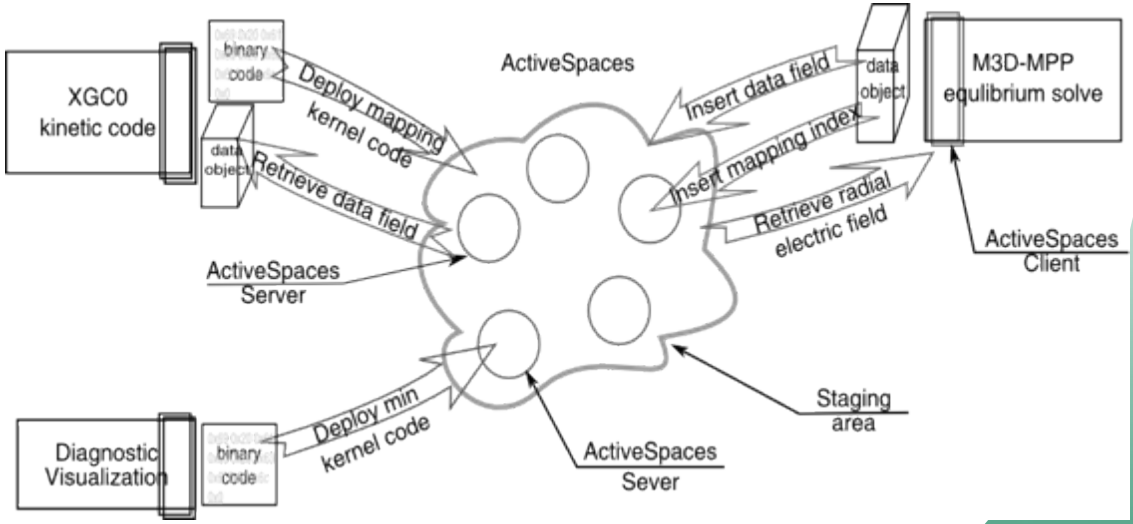


- Staging functions are implemented in C-on-Demand (CoD)
  - CoD can be transparently moved from staging nodes to application nodes or vice versa

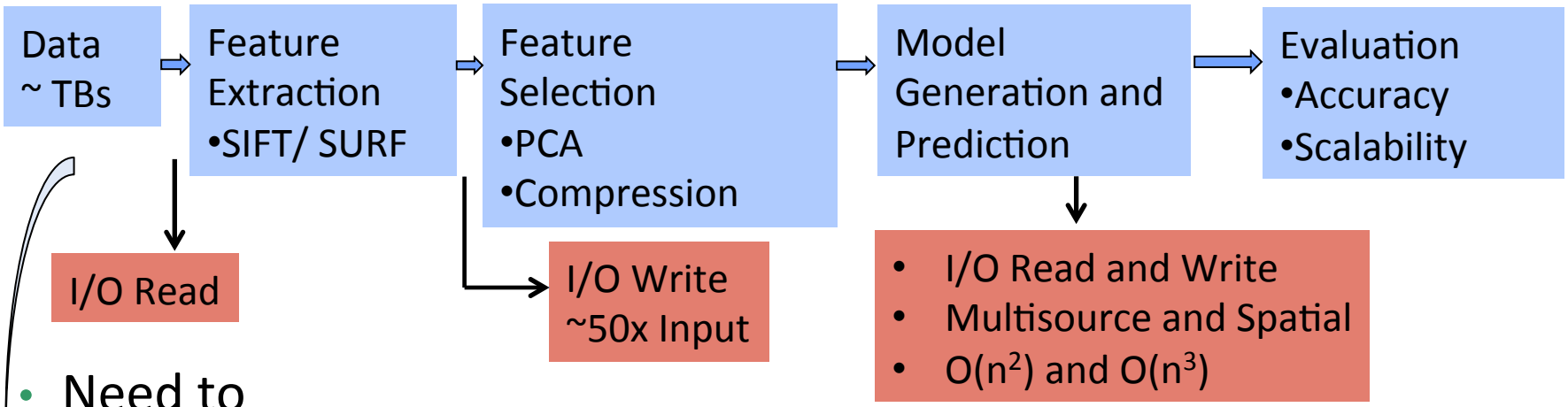
Move work to data (II/II)

- *ActiveSpaces* – Dynamic ***binary code*** deployment and execution
- Programming support for defining custom data kernels using native programming language
  - Operates only on data of interest
- Runtime system for binary code transfer and execution in the staging area





Need in-situ methods for image detection too with state-of-the-art data mining algorithms



- Need to
  - Images processed per hour
    - On clusters and on LCF machines.
  - Accuracy of the prediction
    - Compared against GMM

Data	Task	Sequential	Quad Core	GPU (GTX 285)
One Image (13K x 13K)	SURF	26 min.	3.5x	
45 Images (1K x 1K) (13,000 samples)	GMM (K=20)	120 min.	3x	160x

Source	Dataset Characteristics	Volume
Overhead Images	•Resolution: High (0.6 to 30 m) and moderate (56 m to 1 km) •Training data: Several states in US, parts of Afghanistan and Pakistan	25 TB (image size with features ranges from a GB to TB)
Terrestrial Images	•Small sized photographs: 12 million images •Training data (semantic labels): 2 million images	2 TB (images range from few KB to 0.5 MB)

Conclusions

- ADIOS uses a SOA approach to
  - Get semantic knowledge of the data for code coupling, in-transit visualization, analysis
  - Teams must work together to bridge the GAP between research and production
- Hybrid staging and in situ workflow execution will allow users to intertwine applications, libraries, middleware for complex analytics
- Collaboration is critical to the success of this field
- Now looking for state-of-the-art analytics and visualization routines to be plug-into our system