Distributed Programming Abstractions
Introducing the Theme

Shantenu Jha
Sr. Research Scientist, CCT &
Asst. Research Professor, Comp. Science, LSU
Motivation: In a Nutshell

• Problem Statement:
  • Grid Technology has matured (?); applications that can effectively utilise these technologies are far from ubiquitous!
  • Advances in Grid Applications have not kept pace
    – Number of Applications
    – Novel application

• Computer Science tools, techniques and trends to enable distributed applications
  • Design, implementation and use
  • Dynamic Computing Landscape
  • Evolving Application Requirements
Outline

- Motivation for the Theme
  - Prototype Application: Hurricane Modelling
  - Distributed Applications and Abstractions
- Abstractions: Examples
  - Cactus Framework
  - MapReduce (Google)
  - Simple API for Grid Applications (SAGA)
- Applications: Advantages of abstractions
  - First Principles Grid Application (GridSAT-SAGA)
  - Legacy Application (RepEx-SAGA)
- DPA Theme: People, Goals and Roadmap
Unleashing the power... Computation
Distributed Computing

- Distributed Resources
  - Physically
  - Expertise & end-use
  - Data & control flow
Heterogenous: Data
Source, types, formats, access..

- Simulation data (forecast, nowcast, hindcast)
  - 2D, 3D
- Sensor data (time series)
- Remote imaging
- Aerial photography
- LIDAR
- Weather satellites
- GIS

GOES - 12
MM5 Temp.
MM5 Wind
LIDAR
AERIAL
LANDSAT
MODIS
ADCIRC
Heterogenous: Compute Models

Katrina Forecast Models Legend
Consecutive points 6/12 hours apart:
- GFDL
- CLIP'S
- AIMCS
- NIP'S
- AFUK
- SHIIP
- BAVM
- OFCL NHC Forecast

Ensemble wind fields from varied and distributed sources

Select region and time range
Transform and transport data

Wind Forcing
NCEP
MM5
NCAR
Regional Archives
or
Synthetic Wind Ensembles

Wave and/or Surge Models
ADCirc
ElCirc
WAM
SWAN

Result Dissemination
Archive
Verification
Visualization

Ensemble of models run across distributed resources

Analysis, storage, cataloging, visualization of output
Heterogenous: Coupled Models

Dr. Brian Blanton -- UNC
Programming Distributed Applications
New Challenges

• Characteristics
  • Heterogeneous
    – Resources: systems, platforms, networks...
    – Computational Models, physical models
  • Dynamical Resources
  • Policy Issue
    – Diverging if not contradictory!

• What **should** the end-user focus on? **Must** control? **Should not**? Computational Models?
  – No universal answers!
  – Simplification and distillation
Why this theme matters?

• “We're getting better at predicting hurricanes of 1982”!!
• “90/10 Rule”
  • 90% effort is in setting up and programming the application(s); 10% is left for science, analysis..
• Need to reverse to 10/90
• (some) Ways to lower the barrier
  – Abstractions & Interfaces
  – “Standards Enable Innovation” (Not an Oxymoron!)
Abstractions: End-User Perspective

• Different levels of programming abstraction
  • Programming high-level interfaces
    – APIs e.g, GridRPC, SAGA ..
  • Frameworks
    – Cactus, PetSc, CCA ...
  • Compositional
    – Workflows

Classification not rigid; not mutually exclusive

• Non-programming abstractions
  • Tools (AHE, GridChem...) that abstract resources, resource-mgmt, scheduling, file-transfer etc..
Abstractions: Advantages

- Ease of Application development
  - Separate application development and logic from other “details” of the system
  - Stable and easier deployment on different distributed systems, platforms & architectures
- Effective and efficient utilization of resources
  - Marshalling resources left to the experts, i.e., the “system” via layers below the application
Abstractions: A Balancing Act

• Abstractions are not the solution to all ills!
  • “Death by Abstraction”
  • “… can't always solve a problem, with yet another layer of abstraction”

• Appropriate abstraction is a challenge
  • Abstraction vs performance
  • Abstraction vs flexibility
  • Abstraction vs usability

• Currently for distributed systems – lack of abstractions trumps all other considerations!
Outline

- Motivation for the Theme
  - Prototype Application: Hurricane Modelling
  - Distributed Applications and Abstractions

- Abstractions: Examples
  - Cactus Framework
  - MapReduce (Google)
  - Simple API for Grid Applications (SAGA)

- Applications: Advantages of abstractions
  - First Principles Grid Application (GridSAT-SAGA)
  - Legacy Application (RepEx-SAGA)

- DPA Theme: People, Goals and Roadmap
Cactus: Computational Framework

- Cactus is a modular, portable, manageable environment for collaboratively developing parallel, high-performance simulation
  - Develop modules (components) based upon their expertise
  - No knowledge of the internals or operation of the other modules
  - Modules = thorns = Components
- Framework: An extensible system that is represented by a set of abstract classes and the way their instances interact
Cactus Abstraction: Flesh and Thorns

Cactus Flesh:

- Independent of thorns
- Acts as utility/service library which thorns call for info or to request some action
- Contains abstracted APIs
  - Parallel operations, I/O, checkpointing...
- All actual functionality in provided by thorns

Cactus Thorns:

- Separate libraries encapsulating some specific functionality
- Different thorns can provide the same functionality; interchangable
- Can be written in any language (C, C++, F77..)
Cactus: Flesh and Thorns

Plug-In “Thorns” (modules)
- driver
- input/output
- interpolation
- SOR solver
- wave evolvers

Core “Flesh”
- ANSI C
- parameters
- scheduling
- error handling
- make system
- grid variables

- extensible APIs
- remote steering
- Fortran/C/C++
- equations of state
- black holes
- boundary conditions
- coordinates
- multigrid
- grid variables
- SOR solver
- wave evolvers
- driver
- input/output
- interpolation
- SOR solver
- wave evolvers

CENTER FOR COMPUTATION & TECHNOLOGY AT LOUISIANA STATE UNIVERSITY
Cactus Abstractions: Thorns-Arrangements-Toolkit

- Thorns are grouped into *arrangements*
  - may have related functionality
  - may contain everything needed for one problem

- We call a collection of arrangements, a *toolkit* e.g.
  - Cactus Computational Toolkit
  - Cactus CFD & Relativity Toolkit
  - Cactus Comp Biology Toolkit
Abstractions: Getting it right

- Nothing about distributed computing, so far...
- Integrating Cactus with MPICH-G2; Grid-enabled versus Grid-aware
- Appropriately layered abstraction; A general Grid Framework (extensible)!
- Gordon Bell Prize '01
- Many other frameworks – CCA, Pooma, PetSC

From: Allen et al, Proceedings of SC'01
MapReduce (Google)

- Conceptually simple computation
  - Crawl document, request logs, compute graph structure of web documents etc.
  - Issues of parallelism, distribution, load balancing and fault tolerance convert simple computation into complex code

- Simple, Powerful Abstraction
  - Map & Reduce primitives
  - Reduced Programming Model
  - Hides challenges of distributing and parallelising
  - Construct compound operations (D-sort, D-grep)
  - Optimised low-bandwidth, high-latency networks
Simple API for Grid Applications

Motivation

Q: Why are there so few grid applications out there?

- A lack of simple, stable, integrated and uniform high-level programming interface that provides the most common grid programming abstractions
  
  • Need to hide underlying complexities, varying semantics, heterogeneities and changes from application program(ner)
  
  • Globus job-launch, file transfer etc. exist, but need something that operates at the end-user application level
Copy a File: Globus GASS

```c
int copy_file (char const* source, char const* target) {
    globus_url_t source_url;
    globus_io_handle_t dest_io_handle;
    globus_ftp_client_operationattr_t source_ftp_attr;
    globus_result_t result;
    globus_gass_transfer_requestattr_t source_gass_attr;
    globus_gass_copy_attr_t source_gass_copy_attr;
    globus_gass_copy_handleattr_t gass_copy_handleattr;
    globus_ftp_client_handleattr_t ftp_handleattr;
    globus_io_attr_t io_attr;
    int output_file = -1;

    if ( globus_url_parse (source_URL, &source_url) != GLOBUS_SUCCESS ) {
        printf("can not parse source_URL \\
"%s\n", source_URL);
        return (-1);
    }

    if ( source_url.scheme_type != GLOBUS_URL_SCHEME_GSIFTP &&
         source_url.scheme_type != GLOBUS_URL_SCHEME_FTP &&
         source_url.scheme_type != GLOBUS_URL_SCHEME_HTTP &&
         source_url.scheme_type != GLOBUS_URL_SCHEME_HTTPS ) {
        printf("can not copy from %s - wrong prot\n", source_URL);
        return (-1);
    }

    globus_gass_copy_handleattr_init (&gass_copy_handleattr);
    globus_gass_copy_attr_init (&source_gass_copy_attr);
    globus_ftp_client_handleattr_init (&ftp_handleattr);
    globus_io_fileattr_init (&io_attr);
    globus_gass_copy_attr_set_io (&source_gass_copy_attr, &io_attr);
    globus_gass_copy_handleattr_set_ftp_attr (&gass_copy_handleattr,
                                              &ftp_handleattr);
    globus_gass_copy_handle_init (&gass_copy_handle);

    if (source_url.scheme_type == GLOBUS_URL_SCHEME_GSIFTP ||
         source_url.scheme_type == GLOBUS_URL_SCHEME_FTP ) {
        globus_ftp_client_operationattr_init (&source_ftp_attr);
        globus_gass_copy_attr_set_ftp (&source_gass_copy_attr,
                                       &source_ftp_attr);
    } else {
        globus_gass_transfer_requestattr_init (&source_gass_attr,
                                              source_url.scheme);
        globus_gass_copy_attr_set_gass(&source_gass_copy_attr,
                                      &source_gass_attr);
    }

    output_file = globus_libc_open ((char*) target,
                                     O_WRONLY | O_TRUNC | O_CREAT,
                                     S_IRUSR  | S_IWUSR | S_IRGRP |
                                     S_IWGRP);
    if ( output_file == -1 ) {
        printf("could not open the file \\
"%s\n", target);
        return (-1);
    }

    /* convert stdout to be a globus_io_handle */
    if ( globus_io_file_posix_convert (output_file, 0,
                                       &dest_io_handle)
                                 != GLOBUS_SUCCESS) {
        printf("Error converting the file handle\n");
        return (-1);
    }

    result = globus_gass_copy_register_url_to_handle (gass_copy_handle,
                                                      (char*)source_url,
                                                      &source_gass_copy_attr, &dest_io_handle,
                                                      my_callback, NULL);
    if ( result != GLOBUS_SUCCESS ) {
        printf("error: %s\n", globus_object_printable_to_string
                  (globus_error_get (result)));
        return (-1);
    }
    globus_url_destroy (&source_url);
    return (0);
}
```
Copy a File: SAGA

```cpp
#include <string>
#include <saga/saga.hpp>

void copy_file(std::string source_url, std::string target_url) {
    try {
        saga::file f(source_url);
        f.copy(target_url);
    }
    catch (saga::exception const &e) {
        std::cerr << e.what() << std::endl;
    }
}
```

- Provides the high level abstraction layer, that application programmers need
- Like MapReduce – leave details of distribution etc. out
- Shields gory details of lower-level m/w system
Simple API for Grid Application

Parallel Programming Analogy

Status of distributed programming today, (somewhat) similar to parallel programming pre-MPI days

• SAGA conception & trajectory similar to MPI
  – OGF specification; on path to becoming a standard

• SAGA is to the grid application developer, what MPI is to the parallel program developer (“grid-primitives”)

• Measure(s) of success:
  • Does SAGA enable the rapid prototyping of “new” grid applications?
  • Does it enable enhance functionality using significantly less code?
SAGA API: Design & Specification

• Design and requirements derived from 20 Use Cases (some “correlated”)
  – Different projects, applications and functionality
    • Biological, Coastal-modelling, visualization ..

• Interface is language independent, object-oriented and each sub-system is independent

• Specificed using Scientific Interface Description Language (SIDL)
  – extensible and easily implementable
SAGA: Architecture Overview

SAGA Adaptor development now an OMII-UK project
SAGA: Implementation

SAGA API Packages (managed by 'Engine')

- Files
- Jobs
- RPC

SAGA Adaptors

- Jobs (Globus)
- Jobs (Naregi)
- Jobs (gLite)

Middleware

- Globus
- Naregi
- gLite
SAGA Implementation: Extensibility

• **Horizontal Extensibility – API Packages**
  – Current packages:
    • file management, job management, remote procedure calls, replica management, data streaming
    • Steering, information services, checkpoint in pipeline

• **Vertical Extensibility – Middleware Bindings**
  – Different adaptors for different middleware
    – Set of ‘local’ adaptors

• **Extensibility for Optimization and Features**
  – Bulk optimization, modular design
Outline

• Motivation for the Theme
  • Prototype Application: Hurricane Modelling
  • Distributed Applications and Abstractions

• Abstractions: Examples
  • Cactus Framework
  • MapReduce (Google)
  • Simple API for Grid Applications (SAGA)

• Applications: Advantages of abstractions
  • First Principles Grid Application (GridSAT-SAGA)
  • Legacy Application (RepEx-SAGA)

• DPA Theme: People, Goals and Roadmap
GridSAT
Grid Application Using First Principles

• Satisfiability: *To determine if the variables of given Boolean formula can be assigned such as to make it TRUE*

• Adaptive: computation to communication ratio can be adjustable (!)

• Allows new domain science
  – beats zChaff (time taken and problem)

*Adapted from slides by Wolski & Chakrab*
GridSAT Characteristics

• Parallel, distributed SAT solver
  – Both CPU and Memory Intensive
  – Splitting leads to better performance
  – Allows sharing: clause learned in solver shared

• Grid Aware Application:
  – Heterogenous (single, clusters & supercomputers)
  – Dynamical Resource Usage
    • Unpredictable runtime behaviour
      – How much time? How many resources? When to split? Which process splits first?
  – Problems vary: easy to hard, short to long
    – Need to be adaptive, “add resources as you go”
GridSAT: Programming Requirements

- RPC, Dynamic resource & Job management
- Error Handling, scheduling and checkpointing

SAGA provides the required programming functionality, at the correct level of abstraction and thus makes it easier to manage, deploy and extend (for new functionality)
Legacy Application: Replica Exchange

- “Class of algorithm” used for bio-molecular simulations
  - e.g., Protein (mis-) folding
- Primarily used for
  - Enhanced sampling
  - Determine transition rates
- Task Level Parallelism
  - Embarrassingly distributable!
Replica Exchange Algorithm

- Create replicas of initial configuration
- Spawn 'N' replicas over different machine
- Run for time $t$
- Attempt configuration swap of $R_i \leftrightarrow R_j$
- Run for further time $t$
- ...
- Repeat till termination
RE: Programming Requirements

• RE can be implemented using following “primitives”
  • Read job description
    – # of processors, replicas, determine resources
  • Submit jobs
    – Move files, job launch
  • Checkpoint and re-launch simulations
    – Exchange, RPC (to swap or not)
• Implement above using “grid primitives” provided by SAGA
• Separated “distributed” logic from “simulation” logic
  • Independent of underlying code/engine
  • Science kernel is independent of details of distributed resource management
    • Desktop akin to High-end supercomputer!!
Outline

- Motivation for the Theme
  - Prototype Application: Hurricane Modelling
  - Distributed Applications and Abstractions
- Abstractions: Examples
  - Cactus Framework
  - MapReduce (Google)
  - Simple API for Grid Applications (SAGA)
- Applications: Advantages of abstractions
  - First Principles Grid Application (GridSAT-SAGA)
  - Legacy Application (RepEx-SAGA)
- DPA Theme: People, Goals and Roadmap
Parallel Computing: Redux

• “The Landscape of Parallel Computing Research: A View from Berkeley”
  http://view.eecs.berkeley.edu/wiki/Main_Page

• Multi-core as the route to parallelism. Problems with programming and exploiting parallelism..

• 13 Dwarfs (An algorithmic method that capture a pattern of computation and communication)
  – Actual applications will change; dwarfs are abstractions.

• 7 Questions; Not all answers. But a perspective.

• Post-proposal (Jan '07): Model Document for the output of our theme!
  – “Almost anything is justified, if it makes it easier to write programs that execute efficiently on many core computing”
Framing the Landscape

1. What are the applications?
2. What are the kernels of the application?
3. What are the HW building blocks?
4. How to connect them?
5. How to describe applications?
6. How to program hardware?
7. How to measure success?
The Theme: People

• Core Team:
  • Gabrielle Allen, Peter Coveney, Dan Katz, Thilo Kielmann, Omer Rana

• Workshop I: (30 May – 01 June)
  • Geoffrey Fox, Mark Baker
  • Domenico Talia, Jose Cunha,
  • Murray Cole, Dave Berry, Albert Burger, Dave DeRoure (?) + Invited participants ....

• Workshop II: (Early September)
  • Rich Wolski, Satoshi Matsuoka, John Shalf, Craig Lee, Ian Foster (?), Ian Taylor + Invited ...
The Theme: Roadmap

- **Workshop I**: (30\textsuperscript{th} May - 01\textsuperscript{st} June)
  - Focus on fleshing out the scope, main direction
  - Application “Dwarfs”?
- **Workshop II**: (15\textsuperscript{th} August - 01\textsuperscript{st} October ?)
  - Computer Science Focus
  - Application “Dwarfs” $\leftrightarrow$ Programming Models
- **White Paper writing festivals!**
- **Workshop III**: Jan/Feb (?) 2008
  - Dependent on I & II
- **Satellite Workshop**: “Distributed Applications” - Mardi Gras'08 (LSU) (IEEE/Computer Society)
The Theme: Aims

• Specific Goals:
  • Workshop I and II:
    – Paper merging and updating “Classifying Grid Applications” and “Grid Programming Models”
  • “A view” style report/survey
    – Survey the landscape, scope, primary blocks

• Non-specific Goals:
  – Explore different, interesting and relevant work (Not too prescriptive)
  • Finding common threads
    – “Develop, implement and deploy an application based upon (some) ideas generated by this theme”!
      (Bill St. Arnaud)
The Theme: Your Input

• Topics of relevance that you'd like the theme to address?
• Share your 'distributed' experience?
• Suggestions:
  – Visitors?
  – Speakers?
  – Good Papers?
  – Ideas?
• Tell a friend
• Article to appear in “Grid Today” soon...
Conclusions

- Motivation for the Theme
  - Prototype: Hurricane Modelling
  - Distributed Applications and Abstractions

- Abstractions: Examples
  - Cactus Framework
  - SAGA
  - MapReduce (Google)

- Applications: Advantages of abstractions
  - First Principles Grid Application (GridSAT-SAGA)
  - Legacy Application (RepEx SAGA)

- DPA Theme: Goals and Roadmap
Acknowledgments

• (Core) Members of the DPA theme
• H Kaiser, Ole W, A Merzky (SAGA) & other members of the distributed computing group
• Slides:
  – Cactus: Yaakoub, G Allen
  – Hurricane Modelling: Bogden & Allen
  – GridSAT: R Wolski & W Chakrab

• e-Science Institute