Evolution of the GrADS Software Architecture and Lessons Learned

Fran Berman
UCSD CSE and SDSC/NPACI
Synergistic Research

• GrADS is an ambitious and forward-looking project whose goal is to develop the program development and execution environment required to make performance on the Grid truly accessible for scientists and engineers.

• This requires research and integration of very disparate software into a flexible, robust and coordinated programming environment.
GrADS

• Project has proceeded using phased research and development strategy
  – Integrating mature and evolving software
  – Addressing 1-10 year research problems
  – Focusing on software development for the most complex, dynamic and heterogeneous computational platform to date

• Resulting system (GrADSoft) is more than the sum of its parts
The Basic GrADS Software Architecture
SW Architecture Evolution

• The basic GrADS SW architecture has evolved through the design and development of prototypes for 2 applications:
  – ScaLapack – a stencil-based LU decomposition code
  – Cactus – a modular component-based numerical relativity code

• Both the ScaLapack and Cactus prototypes involved the design and implementation of an end-to-end development and execution strategy

• We have also been expanding and building specific portions of the GrADSoft architecture in addition to the prototypes
The ScaLapack Prototype

- **Focus:** Implement a version of a ScaLAPACK LU library routine that runs on GrADS.
  - Make use of resources the user has at his/her disposal
  - Provide the best time to solution
  - Use GrADS to manage the Grid without the user being involved in the details
ScaLapack Architecture

User → Library Routine → App Launcher → Contract Development → Performance Model

Resource Selector

to Run-time system
The Cactus Prototype

- **Focus:** add GrADS functionality to Cactus
  - Improved resource selection functionality
  - Contract mechanism allows performance-degradation-based migration
  - Evolves distributed services from integrated one-site modules (ScaLapack prototype)
  - *Note:* Different application model (component-based) than ScaLapack (stencil-based)
The Cactus Prototype

Expansion of the basic architecture
Selected Additional GrADS Projects

- Compiler/Resource Selector Interface (Rice and UCSD)
  - Investigating exchange of application-centric information between compiler and scheduler
  - Building prototype of resource selector with appropriate compiler- and user-provided information
Selected Additional GrADS Projects

- Building Grid Applications by Composing Distributed Software Components (Indiana and UIUC/NCSA)
  - PSE - allows user to write composition and execution control script
  - PSE negotiates with GrADS services for resources
  - Instrumented components send performance events to PSE
Selected Additional GrADS Projects

Contract Development
(UIUC+Rice, UIUC+UCSD)

- Have been focusing on
  - how a contract should be defined and monitored
  - how to represent and identify causes of failure
  - what contract violations warrant rescheduling
Work In Progress

- GrADS is in year 2 of a 3.5 year project but there have already been many contributions:
  - Prototype software and end-to-end applications
  - Research on GrADS component interactions and interfaces
  - Research on Contract definition and development
  - Research on Grid economies

- Although the project is not complete, we have already learned many lessons …
Lessons Learned - Applications

- Mature applications are better targets for new software
  - Mature applications for GrADS must be “live”, malleable and have willing collaborators

- **3 first-cut application classes to consider**
  - Big computations that started life as MPI parallel programs and that run on the Grid as a large MPI virtual machine
  - Heterogeneous applications composed from different components
  - Nearly embarrassingly parallel applications (parameter sweep, master worker, peer-to-peer, etc.)

- **The key is structuring the application so that it is effectively decomposable**
Lessons Learned – Application Classes

• “Gridified” MPI Programs
  – Porting an MPI parallel program to a grid environment is much much harder than most people would guess.
    • most MPI parallel programs (such as big PDE simulations) are designed with uniform low latency in mind
    • without serious work on restructuring, variable high latencies kill the performance; you must really work to hide these latencies.
  – The Grid provides a good target for very large-scale problems but performance is hard to achieve
Lessons Learned – Application Classes

- Distributed “component” applications
  - Composing applications which have an interface that can be invoked remotely (e.g. as “application services” by Netsolve, Ninf, CCA, .NET, Corba, etc.) are more easily targeted to the Grid
  - Composing applications that just read and write files is harder
    - you need to write a special grid program that can manage each app and its files and do gridFTPs to move intermediate files around so that the other apps can see them.

- Nearly Embarrassingly Parallel Applications
  - Distributing and getting performance from these applications is more achievable
    - but you still have to worry about the location of shared files, I/O, migration, and adaptation to achieve performance.
Lessons Learned - Configuration Management

- Configuration Management is critical
  - Persistence
    - The software must stand up to more than one use. It can’t be demoware.
  - Version control and coordination
    - Need to ensure that same versions of software are installed (including patches) at all sites
    - Need to install software in the same manner on each platform (e.g., don't manually change directory structures)
    - Need to coordinate what software will be run as root and what will not
    - Need to coordinate the security methods used at different sites
    - Automation of configuration management is essential
  - Documentation of the configuration at each site and consistent maintenance is critical
Lessons Learned - People

“Grid deployment is an exercise in social engineering.”

- **Professional programming staff is fundamental** to achieve success with this kind of a project
  - Integration and configuration management and maintenance not appropriate subject for Master’s and Ph.D. projects
- System admins do not like being told what software should be run as root.
- Design and implementation by committee is a problem, a core set of people to carry out the mission are needed.
- Effective communication is critical-- Email and conference calls are important but they can overwhelm the process.
- **Effective coordination is key**
  - Over 40 people participating in the project: ~12 PIs, 14 researchers, 5 consultants, 9 part-time staff, 0 dedicated managers
Lessons Learned - Research

• Design and implementation of GrADSoft requires substantial vision, research, development, integration, coordination, and robustification
  – Not all of this can be provided by PIs, graduate students and postdocs

• GrADS research problems include 1 year to 10 year problems
  – General research foci:
    • Negotiation
    • Performance modeling and resource selection
    • Adaptivity
    • Extraction and use of application-specific information
    • Simulation
    • Accounting and system-wide behavior (G-commerce, etc.)

• GrADSoft prototypes require above threshold progress on many fronts at once
  – As prototypes become more comprehensive, progress wrt each research problem as well as synergistic progress become more and more important
SW Lessons Learned – the good, the bad, the ugly

- **Software – the good**
  - **Scripting Languages (perl, python, etc.) are good as a way to script complex scenarios on the Grid**
    - Need easy access to Grid/GrADS services from these languages (e.g. need easy way to remotely launch applications and listen for events or access directory services)
  - **Event systems are good**
    - On the Grid, anything can happen, having a uniform and simple event system so that a grid “control program” can listen for and respond to remote application and sensor events is important
  - **Adaptive control is an effective approach**
    - However, time scales are minutes rather than seconds …
  - **MDS-2 and other “virtual organization” tools help SW to be more usable**
SW Lessons Learned – the good, the bad, the ugly

• Software – the bad
  − Integrating multiple pieces of "research grade" software is very difficult
    • We underestimated the effort required …
    • It can be very hard to tell if software packages are working together correctly, even if they appear to be installed correctly.
  − Tools supporting management of secure accounts, executions, and copies across machines in different administrative domains are still evolving
    • GrADS is "ahead of the curve" and because of that, considerable time and effort has gone into creating interim solutions.
  − The integration of older, more stable and widely-deployed versions of software, versus newer, more feature-rich releases adds complexity
    • While we could often make good use of the added features, frequent re-deployment of the 'base' software takes resources away from development of the 'GradSoft' layer.
SW Lessons Learned – the good, the bad, the ugly

• Software – the ugly
  – **Interoperability is fundamental and difficult**
    • Many interoperability problems beyond those currently solved by Globus, etc. (e.g. getting information about installed packages [envt. variables to use], getting accounts from multiple administrators at different sites)
  – **Adaptive resource migration is hard**
    • Must come up with new selection criteria to get new resources
    • Must use intelligence to determine if new set of resources is actually better than old set
    • Things have to be truly out of whack from a performance perspective before migration makes sense
  – **Dynamic optimization is hard**
    • The ultimate role of the dynamic optimizer is to smooth performance so that it more closely matches performance estimates.
  – **The difficulty of managing the software environment increases exponentially with the number of software packages required**
    • We now use MPICHG, Globus, NWS, Autopilot, ScaLapack, …
SW Lessons Learned – the gratifying

Software – the gratifying

- Grid application development and execution works
  - With experts and non-experts
  - With independently developed SW packages
  - With GrADSoft tools
- Adaptivity enables performance-oriented Grid execution
- Grid can be effectively and accurately emulated (MicroGrid)
- SW tools are maturing and can be maintained in coordination as a persistent platform
- Contract mechanism demonstrates that system can self-identify and address problems
GrADSOFT – Final Words

• The GrADS project represents a comprehensive and synergistic effort to build a performance-sensitive, grid-aware program development and execution environment.

• The team has learned how to work together effectively to develop prototypes and research targeted to large-scale, dynamic testbed environments.
  – However 3.5 years is not enough to provide the robust SW needed by the community.

• The next phase of the project will focus on development of GrADSoft at greater levels of integration.