GridRun:

A lightweight packaging and execution environment for compact, multi-architecture binaries

John Shalf
Lawrence Berkeley National Laboratory
1 Cyclotron Road, Berkeley California 94720
{jshalf@lbl.gov}

Tom Goodale
Louisiana State University
Baton Rouge, LA 70803
{goodale@cct.lsu.edu}

Abstract

GridRun offers a very simple set of tools for creating and executing multi-platform binary executables. These "fat-binaries" archive native machine code into compact packages that are typically a fraction the size of the original binary images they store, enabling efficient staging of executables for heterogeneous parallel jobs. GridRun interoperates with existing distributed job launchers/managers like Condor and the Globus GRAM to greatly simplify the logic required launching native binary applications in distributed heterogeneous environments.

Introduction

Grid computing makes a vast number of heterogeneous computing systems available as a virtual computing resource. It is desirable to run native binary programs in order to use these resources efficiently. However, executing native programs in heterogeneous distributed environments typically requires careful staging of the native binary images, complex RSL's or clever job-launcher scripting to select the appropriate executable to run on each hardware platform. Although there are many robust resource selection systems available as an integrated part of Grid schedulers, progress in deploying production metacomputing applications has been hampered as a result of the non-uniformity and inherent complexity of methods employed to mange native code for parallel heterogeneous environments.

Interpreted languages and Virtual Machines are often employed as an abstraction layer that hides architectural heterogeneity [2]. This includes scripting languages, byte-codes, and virtual machines of various forms. However, these solutions have a significant performance impact for compute-bound applications. Native binary programs still offer the most efficient execution environment for compute-bound applications and will continue to play a very important role in distributed applications for the foreseeable future. Therefore, we focus our attention squarely on the issue of simplifying the management and distribution native executables as multiplatform binary packages (fat-binaries).

In order to support a seamless multiplatform execution environment in lieu of virtual machines, we extend the familiar concept of the "fat-binary" to apply to multi-operating system environments. The fat-binary has been a very well-known and successful design pattern for smoothing major instruction-set-architecture transitions within a single Operating System environment. A fat-binary file contains complete images of the binary executables for each CPU instruction set it supports. The operating system's file-loader then selects and executes appropriate binary image for the CPU architecture that is running the program. Fat-binaries have been used successfully for packaging Windows NT programs that could execute at native performance on both DEC Alpha and Intel x86 architectures. In a more widely known example, Apple Computer Inc. used fat-binary executables as an alternative to emulation during the transition from the 680x0 processors to the PowerPC architecture.

From a user's standpoint, the fat-binaries appear to be ordinary program files that execute at native speed on machines with radically different CPU architectures without any additional effort on their part. However, the only available examples of fat-binary execution environments are systems that support different CPU architectures that run the same operating system. We desire this same degree of elegance for binaries that work across multiple Operating Systems as well as different CPU architectures in order to support heterogeneous Grid environments. A Grid-oriented fat-binary execution architecture must support more robust selection criteria than the prior Microsoft and

Apple examples. In addition, some level of compression must be employed to ensure that these "fat-binaries" do not get unmanageably large given the larger number of target platforms it must support. Finally, the file format must work well with existing job launchers for both parallel and distributed environments including Condor, Globus GRAM, GridLab GRMS, and various implementations of MPI [4,7].

In the paper we describe a simple multiplatform execution environment called GridRun that simplifies the creation and management of fat-binary executables for heterogeneous collections of computing resources. GridRun reduces the size of file transfers required to stage the executables, simplifies storage and selection of the correct executable image, and even reduces the complexity of Globus RSL's and Condor submit files for these kinds of jobs. GridRun easily accommodates additional selection criteria that account for heterogeneity in operating systems, instruction sets, and even software libraries.

Related Work

The most typical method for managing binary executables in a heterogeneous environment is to manually stage the executables on each machine. These methods are typically employed when launching parallel MPI-based metacomputing jobs that span multiple sites and computer architectures. Examples of this include the Gordon-Bell winning runtime optimizing transcontinental black-hole simulations performed by Dramlitsch et al In 2001 [9]. MPICH-G2 was used to launch the job on 1500 processors spread across 4 heterogeneous supercomputer systems located on multiple continents (via DUROC), but the executable images had to be manually staged on each of those respective systems. DUROC was also used to provide a separate RSL for the job-launch on each system. Fat-binaries would allow the a single binary image to be staged across the heterogeneous resources as an integrated part of the job launching procedure with considerably less variation in the subjob RSL's.

Some systems manage heterogeneous execution environments using a service model that treats pre-installed native binaries as resources. Platform-neutral RPC interfaces are used to abstract the differences between underlying computing platforms. Systems like NetSolve [6] are typical of an agent-based remote computing service model where the software component stays resident on a server at a fixed location and is invoked via RPC as needed in distributed applications. The remote server paradigm ensures the revision consistency and availability of the software components. However, this very centralized approach provides a rather rigid infrastructure that is essentially out of the user's control. The execution paradigm afforded by fat-binaries places the control of software revisions firmly in control of the user.

VisPortal [11] and the GridLab Information Services [4] provide a looser remote service model where distributed information services (GRIS/GIIS) or local "contact lists" act as central indices for software components that are pre-installed on various machines in their heterogeneous environment. When a job is launched on a particular host, the system queries the information service (eg GIIS) or the contact list to provide the correct location of the executable to the job launcher (eg edit the RSL for a GRAM job launch). However, unlike NetSolve's remote computing model, software components that are indexed in this manner typically only loosely integrated with the information providers used by the MDS, so there is only a weak guarantee that the installed code-revision matches the data presented by the information service. The process of pushing out new code revisions to a large collection of heterogeneous hosts in order to ensure revision control consistency can be tedious and is clearly not scalable.

There are examples of scalable systems using an application-level scheduling paradigm (AppLeS) [8], such as the Application Manager component of the GrADS framework [7] and Nimrod/G [9], where the binaries are moved to computing resources on a demand-driven basis. However, a fat-binary based system provides the same scaling efficiency and code revision consistency without the added complexity of an application-level scheduler or indexing the codes via distributed information services.

Another method for moving native code in a heterogeneous environment is to incorporate sophisticated automation for rebuilding the application from source code as part of its launch procedure. Such systems provide stronger guarantees of code revision consistency. The Cactus Worm [3] exemplifies the kind of application. The Worm is an adaptive Grid application that dynamically discovers additional resources on the Grid at runtime and will migrate itself to "better" resources automatically in response to "contract violation" or other soft resource failures. The Worm's nomadic capabilities depend on Cactus' architecture independent checkpointing mechanism and Cactus' robust ability to automatically rebuild itself from source code on a wide variety of computing platforms. However, when the cost of rebuilding the application from source-code is factored into the performance model employed by the Resource Selector component of the application, it can create a significant barrier to migration. Similar examples can be constructed from the various adaptive application scenarios supported by the GrADS software infrastructure [7]. Migrating the

Worm's executable image in fat-binary form would significantly simplify code management, reduce the costs of remediation for "contract violation" events, and therefore reduce the barriers to migration.

Condor [1] offers the most sophisticated example of a software infrastructure designed from the start to support binary execution in a heterogeneous environment at a system level. The Condor Matchmaker provides a robust architecture for resource selection using boolean comparisons of code requirements to resource attributes that are specified in Classads. After the job is scheduled on a specific resource, the Condor job submit file provides a scripted mechanism for selecting the proper binary from a list of machine executables using file-naming conventions. The main thrust of the fat-binary paradigm described here is to separate the mechanics of this multi-platform binary selection mechanism out into a lightweight standalone tool. These fat-binaries offer considerable benefits for Condor environments by creating a single compact package for the executables that also contains all of the Classads attributes required for the Matchmaker as well as the ability to automatically extract the appropriate executable once the job has been scheduled on a particular machine. But more importantly, the same multi-platform mechanism for managing native executables can can be used across many different job scheduler and brokering system implementations!

GridRun Architecture

The GridRun architecture consists of a file format for storing the multi-architecture binary images and tools for managing the contents of the fat-binary files. By convention, the fat-binary files use a '.run' extension. The fat-binary file format stores compressed binary executable images as well as attributes that describe the execution environment requirements for each image. These attributes include information about the operating system, cpu architecture, and optionally library dependencies.

The 'gridrun' program acts as a file loader for executing the fat-binaries. It encodes a set of attributes that define the characteristics of the runtime environment. Gridrun selects the appropriate binary from the fat-binary file, expands its compressed image, and executes it. The selection of the correct binary image is accomplished using a simple boolean matchmaking mechanism that compares the target machine's attributes to those of each available binary image stored in the fat-binary file. The machine's attributes are built in to the 'gridrun' program at compile-time, but can be overridden at the commandline. The attribute matching system is flexible enough that the user can specify a subset of attributes to use for the matching process or even define custom, application-specific job selection attributes for both the execution environment and the images stored in the fat-binary.

The invocation for GridRun is very similar in practice to using a java virtual machine. You simply type '*gridrun file.run arg1 arg2 arg3'* and the correct binary image will be selected from the fat-binary file (*file.run*), and executed with the specified commandline arguments (*arg1,arg2, and arg3*).

Fat-Binary File Format

The fat-binary images are stored in an HDF5-based file format [12]. HDF5 provides a very flexible, grammatical structure for storing binary data and information about that data (metadata/attributes) in platform-independent binary files. Furthermore, the HDF development team has a consistent track record of high-quality development and support of their software infrastructure on a wide-variety of platforms that spans a decade. We adopted HDF5 in order to leverage their robust platform-independent software infrastructure and thereby minimize the complexity of porting this package to many computing platforms.

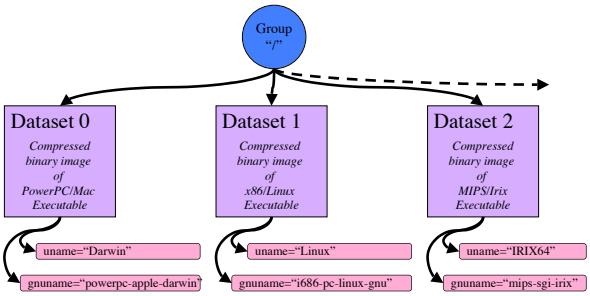

Figure 1: *Schematic of the GridRun fat-binary file format.*

The HDF5 file format is completely self-describing, meaning the file can be interrogated as to its internal structure without any a-priori knowledge of its contents You can think of the internal structure of the HDF5 file as a file system with a nested directory structure. As shown in Figure 1, the root directory level of our fat-binary data schema ("/") is a collection of files (datasets) each of which contains the binary image of a platform-specific executable. Each dataset in the file has a list of attributes associated with it that describe the architecture, software, and operating system characteristics of each executable contained therein. When selecting the appropriate executable from the HDF5 file, one would open the fat-binary file, find the dataset with attributes that match the selection criteria (an efficient random-access operation in HDF5), and then unpack and execute the dataset containing the selected program image.

HDF5 provides an integrated compression system based on the gnu 'gzip' library. Each dataset containing the binary executable image can be packed

using a variable degree of compression. The datasets are compressed individually rather than compressing the entire file in order to maximize the performance of attribute reads required for matchmaking. The dataset must be chunked into equal-sized allocatable blocks in order to take advantage of HDF5's compression scheme. We selected a chunksize of 8192 bytes because it most closely matches typical filesystem blocksizes and because multiplatform tests indicated that it provided the best balance between compression performance and space-efficiency. However, the consequence of this choice is that the compression provides no benefit for executables that are less than 8k in size. The compression capability of HDF5 ensures that the resulting fat-binaries are small enough that they do not create a significant burden when staging files for remote execution, while still supporting efficient file access for selection, extraction, and execution of the native-binary images.

Matchmaking

Each of these datasets can have a list of attributes associated them that act as key-value pairs for various selection criteria. Both the key and the value must be valid ASCII strings. The number of attributes associated with a dataset and their values are arbitrarily extendible, but two attributes are always present – the 'uname' and the 'gnuname'. The gnuname is a three-component description of a computing platform produced by the widely-used 'config.guess' script that comes packaged with the GNU Autoconf system. The three components of the gnu name are the cpu architecture, system vendor, and the operating system respectively contained in an ASCII string separated by dashes. For instance, a Apple PowerPC system running MacOS-X is described by the gnuname string "powerpc-apple-darwin". Therefore, the 'gnuname' provides a very specific selection criterion including operating system, CPU, and even OS version numbers if desired.

By contrast, the 'uname' offers a much looser selection criterion. The 'uname' string is typically generated by the 'uname' command on Unix and Cygwin-based systems. For the MacOS-X system previously described, the uname string is simply "Darwin." These attributes can be manually generated for platforms that do not have the 'gnuname' and 'uname' commands available without loss of functionality. Together, these attributes offer a reasonably robust starting point for matching stored executable images to their respective computing platform.

The attributes associated with an executable in the file can be extended to include a variety of other criteria that are not directly related to the machine architecture. For instance, you can select executables that target the same platform, but use a different MPI implementation. The names of the attributes that define additional selection criteria are entirely up to the user of the fat-binary packaging system. The tools that create and execute this fat-binary file format make it very simple to define an arbitrary number of custom attributes for each binary image.

Implementation

The GridRun system depends on a set of command line tools for managing the fat-binary files. These include tools that execute fat binaries (gridrun,grun) as well as tools for adding (gpack), extracting (gextract), removing (gremove), and listing information (ginfo) about the executables contained in the fat binary file. In fact, all of these tools are soft-links to a single executable that differentiates its functionality based on its invocation name.

The fat binary files are constructed incrementally using the gpack command as described in Figure 2. By default, the file will contain the 'gnuname' and 'uname' attributes, which are built into the 'gridrun' tools at compile time. However, you can define custom additional attributes for the binary at packaging time. For instance, if the executable is built with MPICH G2 and uses software-based OpenGL, it can be specified at packing time using the the '-k' option

gridpack –k mpi=mpich_g2:opengl=mesa -i exename exename.run

In the above example, the arguments after the '–k' option specify additional attributes and their values. These will be associated with the input executable image specified by the '-i' option as it is packed into the 'exename.run' fat-binary file.

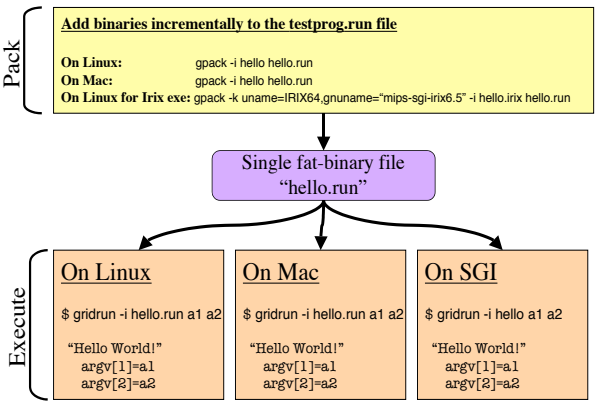

Figure 2: *This diagrams the process for incrementally creating and executing a grid fat binary using the GridRun tools. A single fat-binary (hello.run) is constructed from 3 native binary images. The resulting fat-binary can be executed natively on all of the platforms.*

Likewise, you can encode attributes that have exactly the same names as Condor Classads architecture descriptions in order to simplify interoperation with that environment. The 'gnuname' and 'uname' are always present as attributes, but they can be overridden. For instance, when packing binaries for multiple platforms resident on the same machine, the 'gnuname' and 'uname' can be specified explicitly for each of those binaries as they are packed.

The 'ginfo' command lists the contents of a fat binary file as a plain-text summary. The listing includes both the number of stored binaries and the attributes associated with each binary. Currently, the listing appears as a simple indented list of items, but eventually will be able to output that attributes in Classads syntax for closer integration with Condor or as an XML markup to support integration with web services.

The 'gridrun' (or 'grun') command selects an appropriate executable using the matchmaking mechanism specified earlier in this paper and executes the extracted binary. The default mode of operation is to select based on the 'gnuname' of the system, but this can be overridden using any set of matching criteria using commandline switches. For instance;

gridrun –k uname file.run

Will cause gridrun to use the 'uname' attribute for matching instead of the 'gnuname'. When a value for the attribute isn't explicitly provided, it will default to the internally stored value of the attributes that are built into the 'gridrun' command. In order to select a specific binary implementation, the 'uname' can be specified explicitly.

gridrun –k uname=Linux file.run

You can even define a list of matching criteria explicitly. For example;

gridrun –k uname,mpi=mpich_g2 file.run

machine	gnuname	uname
Linux on x86	i686-pc-linux-gnu	Linux
Linux IA64	ia64-unknown-linux-gnu	Linux
NEC SX-6	sx6-nec-superux	Super-ux
SGI Onyx	mips-sgi-irix	IRIX64
IBM SP2	powerpc-ibm-aix	AIX
Apple G5	powerpc-apple-darwin	Darwin
HP Workstation	hppa2.0-hp-hpux	HP-UX
Sun Workstation	sparc-sun-solaris	SunOS

Table 1: GridRun has been tested on the following systems. Support for Windows is coming soon, but was not ready in time for this paper.

first matches against the 'uname' attribute of the binary executables and then matches the 'mpi' attribute to ensure that an mpich_g2 implementation is chosen. Currently, if those attributes are not specified in the description of the binary file's characteristics, they will not be factored in to the selection. However, they become a requirement if those attributes are present in the description of the stored binary image.

Normally Gridrun will create a temporary copy of the binary and then remove that temporary image when execution is completed. However, there are situations where the binary image needs to remain resident. In that situation, the 'gextract command will unpack the executable using the same matching machinery as '*gridrun*,' but will not execute the binary or remove it once the command is completed. The command, '*gremove'*, deletes a binary image from the fat binary file.

Thanks to the elegance of the HDF5 file format, the entire package is extremely easy to port to a wide variety of computer systems. Table 1 lists the currently supported systems. Windows support is coming soon, but was not ready at the time of this writing.

Considerations for MPI

Gridrun has also been modified to support a variety of parallel job launchers. OpenMP and other threaded implementations are supported trivially, but the implementation details of various MPI job loaders require special consideration. For instance, if you invoke a grid fat binary with mpich, the 'Gridrun" command will be executed simultaneously by each processor.

mpirun –np 8 gridrun myexe.run myargs

On a shared filesystem, the job launch results in 8 concurrent extraction requests – 7 of which are redundant. While these redundant extractions do not generally cause the job-launch to fail, their primary impact is to slow the job launching process significantly with a flood of I/O activity. The '-1' option causes gridrun to only extract the binary if one is not already present. A fine-grained file locking mechanism is required guarantee that redundant extractions do not occur, but such mechanisms are extremely non-portable. So, while it is not possible to entirely eliminate redundant extractions, the MPI processes are typically started with sufficient time-skew that redundant extractions rarely occur in practice despite the lack of synchronization.

Similarly, when executing a parallel job in a heterogeneous environment that has a shared

filesystem, it may be necessary to have the names of the temporary executables be differentiated in order to prevent collisions. The '-h' option (for heterogeneous) provides this functionality by appending the 'gnuname' of each executable to the name of the executable. More flexible naming options will be supported in the future.

Eventually the detection of MPI job launchers will be automated, thereby eliminating the need for these platform specific flags. However, we are still in the process of validating the current set of methods in order to ensure they cover the vast majority of job-launcher implementations. Therefore, many of these context-dependent flags will disappear as GridRun matures

A Simple Condor Use-Case

A Condor multi-architecture submit file will typically provide a 'Requirements' specification that specifies multiple machines using an '||' (or) clause. For instance

Requirements = ((OpSys == "LINUX") ||
(OpSys == "WINNT"))

The "Executable" is then specified with an extension that corresponds to the computing platform it can run on.

Executable=hello.$$(OpSys)

When the Condor matchmaker selects and schedules the job on a specific host, the definition of the "Executable" name in the submit file constructs the correct executable name. For instance, if scheduled on a OpSys=LINUX machine, it will select hello.LINUX and if scheduled on a machine with OpSys=NT, it will select hello.WINNT. A considerable amount of name mangling must be applied to the executable file for more complicated selection scenarios.

When used with GridRun, the 'Executable' specified in the Condor job submit file will always be the full path to the 'gridrun' program where it is installed on the remote host regardless of the its architecture. The first argument specified for the 'gridrun' executable will be the name of the fat-binary file ('hello.run' for this example) that has been pre-packed with executables for each machine type allowed by the 'Requirements' specification in job submit file. The job submit file must use the 'transfer_input_files' directive to ensure that 'hello.run' fat-binary file is transferred to the remote host instead of 'gridrun' (the default is to transfer the program specified as the 'Executable'). When the job is scheduled on a remote host, 'gridrun' will unpack and execute the correct binary program contained 'hello.run' executable for the selected computer system. No mangling is required for the name of the executable and the user only needs to keep track of a single executable file 'hello.run'.

A Simple Globus Use-Case

In order to use GridRun with Globus, the 'gridrun' tools need to be installed in a well-defined path on all hosts. The most appropriate location is in the same path as the Globus commands in order to ensure consistent operation.

Normally, an RSL is used to specify paths to the executable and to stage data files for the job when launching a remote job with 'globusrun'. If the executable is pre-staged to the remote hosts, the user must know a-priori the proper path to those executables before executing 'globusrun'. Otherwise, if the file must be staged by 'globusrun,' the user must know a-priori the architecture of the remote host, select the correct executable, and specified either as part of the 'execution' URL or using the 'file_stage_in' parameter to transfer the file via the GASS server. Either approach can be quite difficult in when using 'globusrun' with to launch jobs on heterogeneous collections of computers.

When using 'globusrun' to execute a 'gridrun' fat-binary, the RSL will always specify the full path to 'gridrun' as the executable. In addition, the 'file_stage_in' directive is used to copy the user's local fat-binary executable over to the remote host for execution. The fat-binary must be pre-packed with the executable images for any of the remote computer architectures the job might be submitted to. The GASS server can then be used to stage the fat-binary to the remote host as part of the submission, and the fat-binary file is specified as the first argument to the 'gridrun' command executed on the remote host. The 'gridrun' will unpack the correct executable from the fat-binary and execute it. Gridrun allows a user to employ a single generic RSL to make job submissions to a heterogeneous collection of computers.

Performance

Multi-platform "fat" binaries will greatly simplify the distribution and management of code in heterogeneous grid environments, but these benefits will be rendered moot if the system adds too much overhead to the process of distributed job launching. Here we quantify the sizes of the fat-binary files and infer their effect on distributed job-launching performance?

Figure 3 shows the file sizes of a large binary executable built for 2 different platforms: i686-pc-linux-gnu and powerpc-apple-darwin. The combined size of all the two images stored in the fat-binary is actually slightly smaller than either of the original

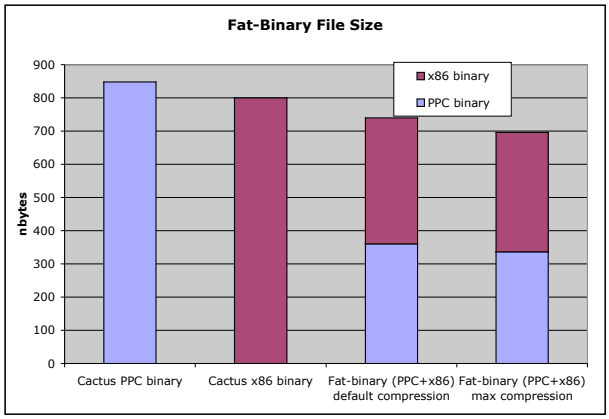

Figure 3*: A GridRun fat-binary file containing binary images for two different executables (one for MacOS-X PowerPC and the other for a Linux x86 workstation) is smaller than either of the original executables. In this case, the binary image is of the Cactus code, a complex simulation code that evolves Einstein's equations in 3D.*

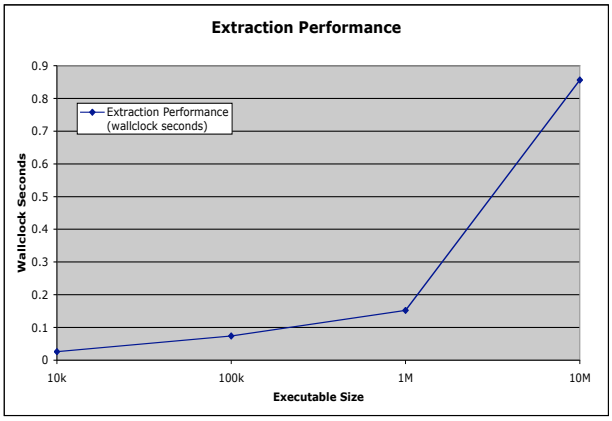

Figure 4: *The extraction of the correct binary image from the fat-binary file was found to dominate the gridrun execution time. In order to test the extraction performance, artificial binary files of various sizes were constructed using a randomly generated data. The test platform in this case was a Mac G4 notebook computer (the slowest platform).*

executables. More aggressive compression can be specified during the packing process, but it rarely offers significant benefits. Therefore, we conclude that the fat-binary file format will not significantly increase the file-staging overhead when storing a moderate number of executable images.

The selection, extraction, and execution of a binary image from the fat-binary file adds some overhead to the job-launching procedure. However, for most executables, it will likely amount to a fraction of a second latency in startup time for a job. You can see in Figure 4 that while the extraction performance rises rapidly as the size of the executable file increases, it still only accounts for less than a second of the job launching time for executable images exceeding 10 megabytes in size. The Cactus code, for instance, is a very large and complex code, but its executable image is typically less than 5 megabytes in size. Likewise, studies of the performance of the matchmaking mechanism found that the time spent selecting an executable among 20 stored images was less than 0.01 seconds – a testament to the efficiency of the HDF5 file format.

Future Directions

The current implementation has focused on ensuring the GridRun binary selection and execution mechanisms work properly in a heterogeneous environment. Continuing development will focus on improving the integration of this package with existing Grid services including the Globus GRAM as well as resource brokers for heterogeneous Grid environments like Condor, the GrADS Application Manager [7], and the GridLab GRMS[4]. For instance, the current 'ginfo' command can be modified to output information in Condor Classads format or automatically generate a job description file suitable for Condor or GRMS. The fat-binary could also store performance models or historical performance data with each executable in order to support some of the application scheduling logic of the GrADS framework. The fat-binary file contents could also be expressed in XML form in order to simplify integration with web services. Even the gridrun command itself could be a proxy to Condor, GrADS, or GRMS services. Merely invoking a gridrun command on a multiplatform binary could initiate a search for appropriate grid resources on which to actually execute the command. Like all middleware, gridrun will be most useful when integration with existing services renders it nearly invisible.

There is also considerable interest in packaging multiplatform dynamically loadable objects. This capability will be important for frameworks based on the Common Component Architecture (CCA) [10] that assemble applications from modular components packaged as dynamically loadable objects. We can extend the component loading technology in existing CCA frameworks to use gridrun technology to inspect a multiplatform binary component package and dynamically load the appropriate component image.

Finally, we would like to find ways to automatically create multiplatform binaries at comple-time. Recent developments in the gnu cross-compiler environment enable multiple cross compilers to coexist in the same space. One could theoretically create a compiler environment that orchestrates multiple cross-compilers to generate multiplatform executables in a single step compilation process. While gcc is no

longer well regarded for producing efficient executables, such a system would offer a functionally complete multiplatform native code generation environment. This kind of single-pass multi-platform compiler technology can offer more uniformity in the build environment than is currently possible using multiplatform makefiles, and would certainly speed the creation of gridrun fat binaries.

Conclusion

Gridrun provides a conceptually simple framework for executing native programs in heterogeneous computing environments. GridRun successfully moves code selection mechanisms that are typically deeply embedded in distributed resource management frameworks and recasts that capability in the form of a lightweight standalone package providing a simple and uniform automation for managing native binary executables in heterogeneous collections of computer systems. The resulting architecture makes execution of native binaries no more difficult than using a virtual machine like Java. The code base has been ported to a wide variety of computing platforms. Thanks to a compact and robust file format, Gridrun executes efficiently in a distributed environment and will not significantly degrade job-launching performance. The result is a uniform job description and execution mechanism that can be used broadly in many different distributed job management and resource brokering applications.

Bibliography

[1] R. Raman, M. Livny, and M. Solomon. "**Matchmaking: Distributed resource management for high throughput computing**." In Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, July 1998.

[2] Ian Taylor, Matthew Shields and Ian Wang. **"Resource Management of Triana P2P Services"** Grid Resource Management, edited by Jan Weglarz, Jarek Nabrzyski, Jennifer Schopf and Maciej Stroinski. (*To be published by Kluwer*,) June 2003. (http://www.gridlab.org/Project/Publications.html)

[3] G. Allen, D. Angulo, I. Foster, G. Lanfermann, C. Liu, T. Radke, E. Seidel, and J. Shalf. **"The Cactus Worm: Experiments with Dynamic Resource Discovery and Allocation in a Grid Environment."** International Journal of High Performance Applications and Supercomputing 15(4), Winter, 2001.

[4] Gabrielle Allen, Kelly Davis, Konstantinos N. Dolkas, Nikolaos D. Doulamis, Tom Goodale, Thilo Kielmann1, André Merzky, Jarek Nabrzyski, Juliusz Pukacki, Thomas Radke, Michael Russell, Ed Seidel, John Shalf and Ian Taylor. **"Enabling Applications on the Grid: A GridLab Overview."** International Journal of High Performance Computing Applications: Special issue on Grid Computing: Infrastructure and Applications, to be published in August 2003.

[5] R. Buyya, D. Abramson, and J. Giddy, **"Nimrod/G: An architecture for Resource Management and Scheduling System in a Global Computational Grid,"** The 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000).

[6] Agrawal, S., Dongarra, J., Seymour, K., Vadhiyar, S. **"NetSolve: Past, Present, and Future - A Look at a Grid Enabled Server, Making the Global Infrastructure a Reality,"** Berman, F., Fox, G., Hey, A. eds. Wiley Publishing, 2003.

[7] D. Angulo, R. Aydt, F. Berman, A. Chien, K. Cooper, H. Dail, J. Dongarra, I. Foster, D. gannon, L. Johnsson, K. Kennedy, C, Kesselman, M. Mazina, J. Mellor-Crummey, D. Reed, O. Sievert, L. Torczon, S. Vadhiyar, and R. Wolski. "**Toward a Framework for Preparing and Executing Adaptive Grid Programs**." IPDPS, 2002.

[8] F. Berman, R. Wolski, S. Figueira, J. Schopf, G. Shao**, "Application-Level Scheduling on Distributed Heterogeneous Networks,"** Proceedings of Supercomputing '96, 1996.

[9] G. Allen, T. Dramlitsch, I. Foster, N. Karonis, M. Ripeanu, E. Seidel, and B. Toonen. **"Supporting Efficient Execution in Heterogeneous Distributed Computing Environments with Cactus and Globus."** Proceedings of Supercomputing 2001.

[10] Rob Armstrong, Dennis Gannon, Al Geist, Katarzyna Keahey, Scott Kohn, Lois McInnes, Steve Parker, and Brent Smolinski**. "Toward a common component architecture for high performance scientific computing.**" In Proceedings of the 8th High Performance Distributed Computing (HPDC'99), 1999.

[11] C. Siegerist, P. Shetty, W. Bethel, T.J. Jankun-Kelly, O. Kreylos, K.L. Ma, J. Shalf. **"VisPortal: Deploying grid-enabled visualization tools through a web-portal interface."** Wide Area Collaborative Environments Workshop, Seattle, WA., June 2003.

[12] http://hdf.ncsa.uiuc.edu

[13] *http://vis.lbl.gov/Research/GridRun*