Porting LCG Software to IA64

Stephen Eccles (technical student at CERN openlab), April 2004

1. Introduction

This document describes work done in order to support the first installation of the software for an LHC Computing Grid (LCG) node on a cluster of IA64 machines. Previous nodes had all been installed on IA32 architecture machines. This work was carried out within the OpenLab research group, IT Division, CERN between July 2003 and February 2004.

This document covers the issues involved in making the necessary software available for installation of an IA64 node. It does not cover the actual work done to install and configure the node and to get it up and running.

1.1 Why this task was necessary

IA32 binary versions of all the required software for LCG Grid nodes exists and is actively deployed on working nodes. In principal, this IA32 software would install and run on IA64 machines. The primary motivation for carrying out this task was therefore to be able to run binary versions of the software specially compiled for IA64 on the IA64 machines so as to be able to assess the optimal performance of this code on IA64 machines.

This choice was forced to some extent anyway because the standard method for installing an LCG node is to use the installation tool LCFG[1], which requires that a specific x86 version of Red Hat Linux (version 7.3) be installed as a base. This same version of Red Hat Linux is not available for the IA64 platform.

Software for LCG is only distributed in binary form and no version for IA64 is currently distributed. Therefore, the task required was to re-compile all the required software on the IA64 platform, making modifications to source code and build/configuration files to port these to IA64 as necessary.

It should be noted that at the time of writing, the LCG software distribution was based on distributions of European DataGrid (EDG) software[2]. The EDG project has now finished and so the responsibilities for that stage of the work will have to change. The work discussed in this document, refers to the production of EDG software for LCG as that was the supported scheme at the time this project was carried out.

1.2 Overview of the structure of this document

Chapter 2 gives an "Executive Summary" of the work carried out and the issues that arose whilst carrying out the task.

Chapter 3 gives an overview of the software that it was necessary to port and re-compile.

---

[1] Local ConFiGuration system, developed and maintained by University of Edinburgh and selected as the supported installation method for EDG and LCG binary distributions.

[2] Produced as part of the EU European DataGrid (EDG) project [1].

Chapter 4 describes the work done to port and compile the Virtual Data Toolkit (VDT) software that is required as a base for all LCG Grid nodes.

Chapter 5 describes the work done to port and compile the required EDG software components that are required for LCG nodes.

Chapter 6 describes the issues involved in acquiring IA64 versions of the other necessary external software components that were required.

Chapter 7 gives a summary of the main issues that were encountered whilst carrying out the above tasks.

Chapter 8 gives suggestions for how VDT and EDG binaries for LCG could be (more easily) obtained for IA64 in future.

Finally, some conclusions are given in Chapter 9.

2. Executive Summary

The task discussed in this document was to take the existing available source code used as a base for an LCG node running on the IA32 architecture and make the equivalent software available so as to be able to have an LCG node running on machines with IA64 architecture with binary code compiled for IA64.

This chapter gives an overview of the work carried out and the main issues arising from it. These points are discussed in more detail in the subsequent chapters 3 to 9.

2.1 Work carried out

The work carried out can be broken down into 3 main components, which are: -

1. Producing IA64 binaries of the Virtual Data Toolkit (VDT) software[3].

2. Producing IA64 binaries of the EDG software necessary to set up an LCG node.

3. Acquiring or producing external supporting software required for the installed system and during the build process for steps 1 and 2.

2.2 Summary of outcome

The binary software (in RPM format) required to set up a running LCG node running on IA64 machines was provided. This was subsequently successfully installed and tested on an LCG test node. The software provided was sufficient to set up a Worker Node, Storage Element and Computing Element running IA64 binaries.

A reproducible build procedure was developed for building VDT IA64 RPMs from source using 2 line commands (one to set the environment, one to build). Once they were discovered, the simple to use build methods for EDG software as supplied with the source code were used with minor modifications.

2.3 Issues encountered during porting of VDT

An issue commonly encountered throughout the task with both the was that of determining the method of how to build the software from source. The majority of the time, these were not an issue related to building on IA64 but were in fact largely due to the factors below: -

1. VDT software is usually only distributed in binary form. Only members of the VDT development teams actually build the software from source generally. Therefore, the procedures and build environment settings required for building from source are not fully documented for the benefit of others wishing to do this, although some basic documentation was available.

[3] Produced and distributed jointly by the New Middleware Initiative (NMI) group and University of Wisconsin – see section 4.1 for details.

2. Once it was ascertained, the prescribed build method failed to work. The Grid Packaging Toolkit (GPT) information files that were distributed with the source code had not been maintained in conjunction with the source.

3. Some specific features that were not obviously documented led to time-consuming experimentation to reach required configurations (e.g. use of Globus "flavors").

2.3.1 Issues encountered with porting EDG software

Almost all of the necessary work to the source code and build system to produce IA64 binaries had been made by the developers. No major porting issues were encountered with the EDG software itself. However, it was time-consuming to get the first versions of this software to build. This could have been avoided with a small amount of extra work within the development teams.

1. As with VDT above, the procedure and build environment required for building from source was not clearly documented. However, once the correct procedure was worked out, the build completed smoothly and was easily reproducible with later versions of the EDG sources.

2. The IA64 build path had not been fully maintained. Some builds failed with errors. These errors were generally simple to fix once familiarity with the source code structure and build procedure had been gained.

Once these issues had been addressed, and the supplied procedures have been straightforwardly repeatable with later versions of the1 source code.

2.4 Conclusions

Almost all the work in this project involved changes to build configuration files and providing the right supporting environment of external RPMs and dependencies and environment variables. Most necessary changes had generally been made to build procedures and files by the appropriate development teams. These modifications had not been maintained in a few cases, requiring minor modifications to fix these problems but no significant "new" porting of build files was necessary.

Almost all the source components of software used were compilable on IA64 without further modification. A significant number of non-critical compiler warnings were received indicating that some cleaning up of the code to optimise it for IA64 would be desirable.

Most of these issues were not specific to IA64 as such but would have occurred if trying to use the build process for any platform other than the one routinely used by the development teams (i.e. IA32).

A reasonable suggestion would seem to be that the VDT and EDG development teams incorporate the small changes required to make building for IA64 straightforward and either make IA64 distributions available or distribute tested buildable sets of source code and build files.

3. More detailed overview of the task and software components needing to be ported

Figure 1 shows the stack of software required to be installed on an IA64 LCG node and compares this with the software installed on an IA32 LCG node.

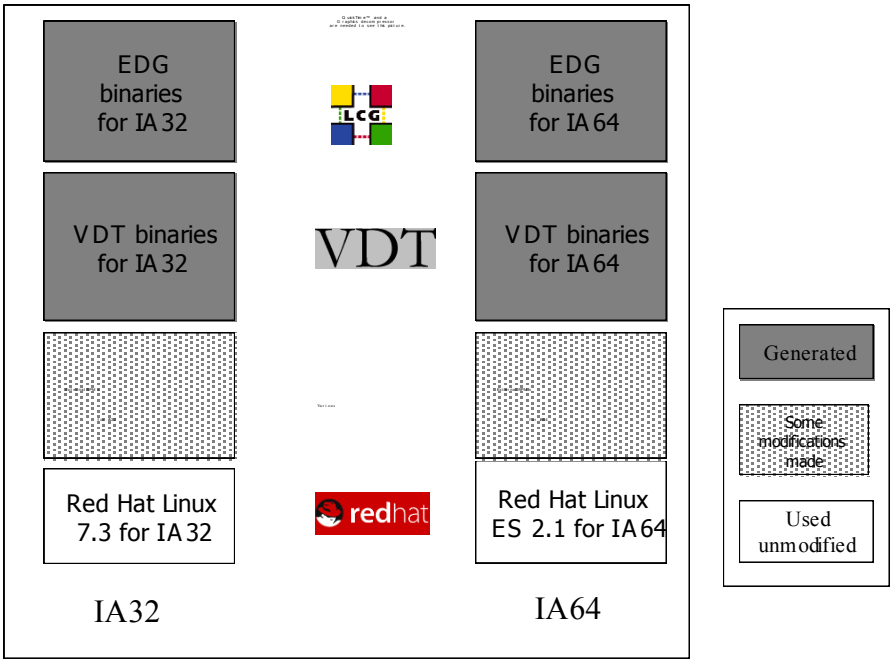


Figure 1: Comparison of software required for IA64 and IA32

3.1 Red Hat Linux Operating System

The (IA32) EDG distribution at the time of writing was stated as being supported only on the Red Hat Linux operating system, version 7.3. This version was not available for IA64. The version installed on the IA64 machines was Application Server (AS) version 2.1 for IA64 which contained earlier versions of many RPMs compared to version 7.3 for IA32 (as well as there being some differences between the set of RPMs in the distribution).

3.2 External RPMs

External RPMs are defined as those that are distributed publicly within by the Linux community but do not form part of the Linux operating system distribution. Examples of external RPMs that were required were specific compilers (e.g. javac), build tools (e.g. ant) and support tools (e.g. boost, classads). These external RPMs fell into two main categories: those used unmodified as distributed publicly and those modified (patched) for EDG/LCG.

There were some issues regarding locating external RPMs at the required versions. These issues are discussed in chapter 6.

3.3 VDT binaries

For IA32, binaries for VDT are distributed by a dedicated team at the University of Wisconsin in the US. At the time of writing, no binaries were currently distributed for the IA64 platform. A version of the VDT source code was available from which the binaries could be built (although this service appears to have been withdrawn recently). There were several issues with converting this source code into suitable binary RPMs for IA64. These issues are discussed in more detail in Chapter 4.

3.4 EDG binaries for LCG

For IA32, EDG binaries for LCG are distributed by a dedicated team based at CERN. At the time of writing, no binaries are currently distributed for the IA64 platform. Source code was available with an associated build procedure for producing RPMs from these sources. There were some issues encountered with doing this and these are described in more detail in Chapter 5.

An overview of the work that was required to be done and their dependencies is shown in Figure 2.

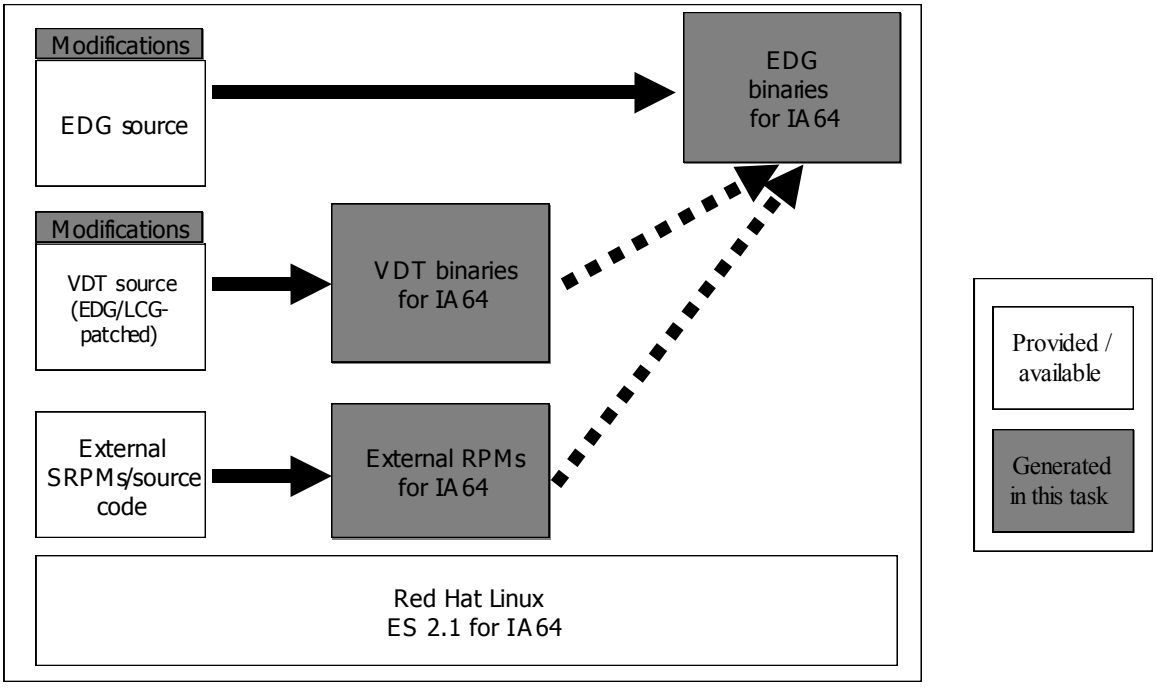


Figure 2: Overview of tasks required

4. Producing IA64 binaries of VDT

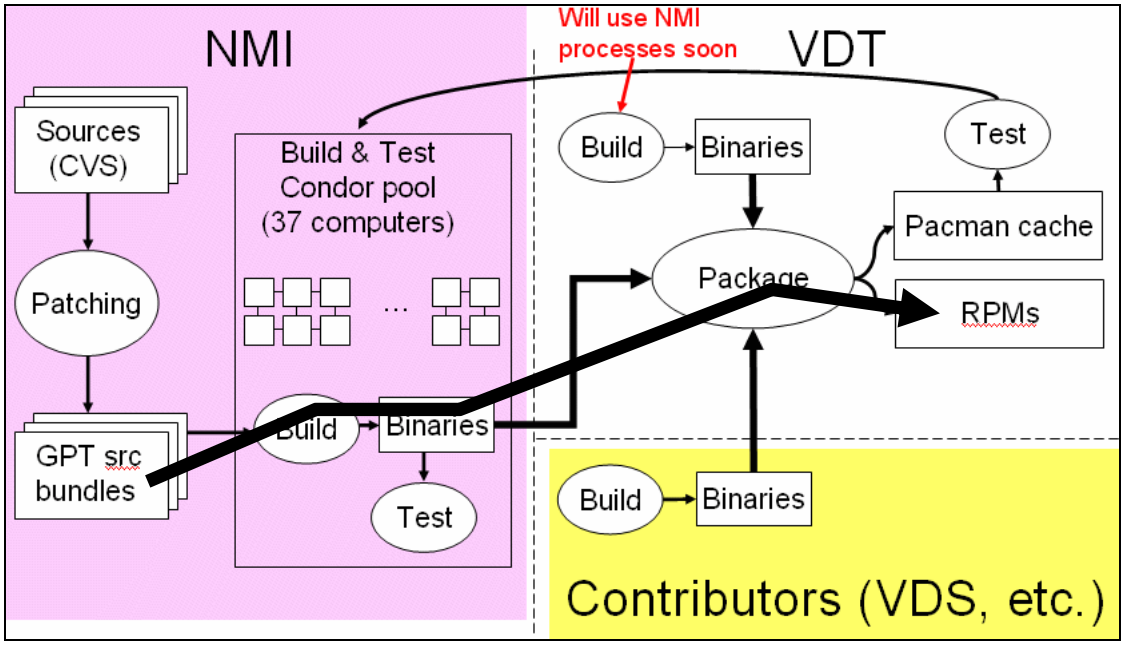
This chapter introduces VDT in more detail and describes the issues encountered in compiling an IA64 version from the source code bundle supplied and then converting the compiled binaries produced into the RPM form desired to perform an LCG node installation.

4.1 Introduction: what is VDT?

"The Virtual Data Toolkit (VDT) is an ensemble of grid middleware" packaged together in to make it easy "to make it as easy as possible for users to deploy, maintain and use" [2]. It includes Condor-G and Globus.[4]

This software is maintained and built by supplied by the New Middleware Initiative (NMI), a group of the National Science Foundation (NSF) and the Computer Science departments of the University of Wisconsin at both Madison and Milwaukee (UWM). UWM distribute VDT in binary form via the VDT web site [1]. The source code used as a basis for the porting work was obtained from this same site.[5]

Figure Figure 3 shows the full VDT build procedure used by NMI and the VDT team to produce the official VDT binary distribution and the set of taks that were required to be carried to produce the IA64 installation as described in this document.



(Figure from VDT web-site [1])

Sub-tasks required to be done for IA64

[4] VDT was based on Globus version 2.2.6 when this project was carried out – the current version of VDT is based on Globus version 2.4.3.

[5] At the time that the work for this project was carried out, the source code for VDT was also made available from this site but this service appears to have been withdrawn recently.

Figure 3: Overview of build process used to produce VDT binary distribution

The EDG software requires modifications to the Globus derived components of VDT. At the time this task was carried out, a separate modified version of VDT was distributed for EDG/LCG. The main features of this version were that it: -

- Was patched specially for EDG / LCG
- Contained a subset of full VDT
- Was packaged as RPMs (rather than using PacMan as is used for standard VDT)

Since VDT version 1.1.13 released 2nd March, 2004, the patches to VDT made for EDG and LCG have now been incorporated into the main VDT release. Therefore, there is no longer a separate patched version of VDT released for EDG/LCG. According to the VDT website [2] "many of these patches have been integrated by the Globus team and will be in a future release of Globus."

The steps required to build the VDT binaries are summarised the diagram below.

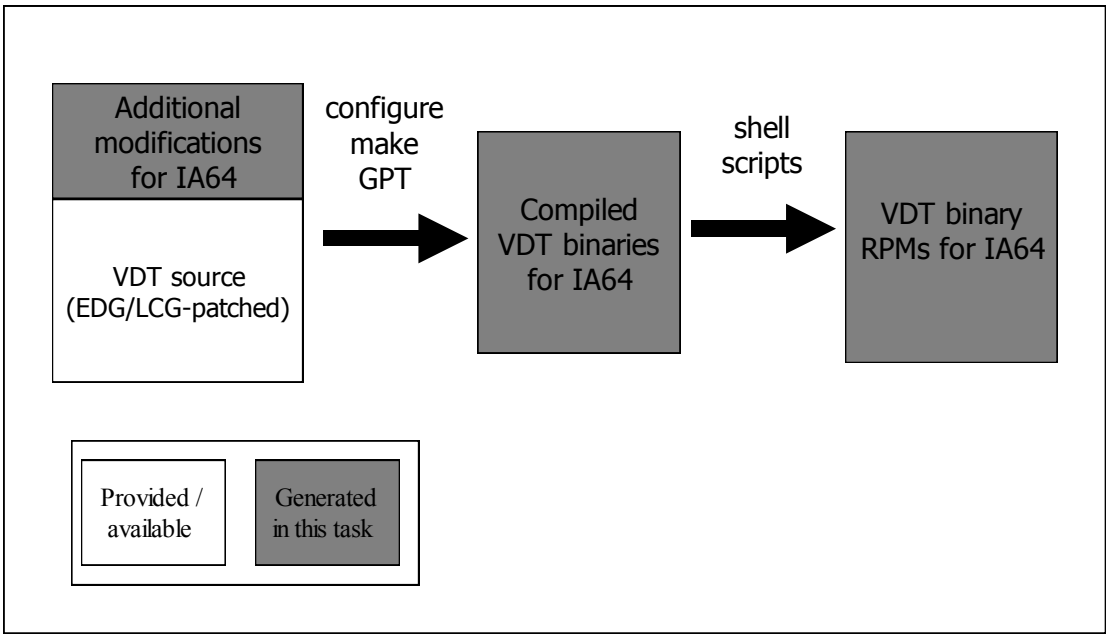


Figure 4: Steps required to build VDT for IA64

4.2 Build method as supplied by VDT

The VDT source code tree was designed to be built with the Grid Packaging Toolkit along with the use of Globus flavors. (These terms are discussed in more detail below.) VDT is however only officially distributed in binary form. The documentation on the VDT website is therefore aimed at the user wishing to install and configure a VDT installation from these distributed binaries. However, some

basic instructions for building VDT are contained within files supplied with the VDT source tarball.

The main problem encountered at this stage was that: -

- The patching of the VDT sources done for EDG/LCG meant that the documented build instructions for VDT (using GPT) failed to work.

The most time-consuming issue that needed to be dealt with as a result of this was that: -

- Building components in the correct order in order to satisfy dependencies between VDT components (see section 4.3 below).

In addition, some other issues were encountered: -

- Building components with the correct required "flavors" (see section 4.2.2 below) was necessary. It was not immediately obvious which flavors were required for which components, so this took some trial-and-error. Some components required a generic "noflavor" flavor to be built, for example, otherwise later component builds failed.

- Following the build instructions results in the compilation binaries and optionally their installation on the same system. It was required for this project to build binary RPM packages that could be used to install the software on other machines. This a further mechanism was required to be developed to do this (see section 4.5).

4.2.1 Grid Packaging Toolkit (GPT)

The Grid Packaging Toolkit (GPT) is a set of tools that unpack, build, link, and install Globus from suitably packaged source code. GPT also provides tools for querying the configuration of a Globus installation.

For building, GPT uses package information files (written in XML) that are provided along with the source code in "bundles". Line command options to these commands are used to specify configuration options such as the flavor that is used for the build.

4.2.2 Globus flavors

The concept of flavors was introduced with the Globus Toolkit. Flavors are defined against specific features of the libraries, daemons, and tools, such as support or lack of it, for POSIX-threads, debugging, and GNU GCC or vendor compiler. By specifying flavors in configure options, a set of such options can be selected.

Specific flavors are provided 64-bit architectures in the Globus Toolkit and thus VDT.

4.2.3 Why GPT would not build the EDG-patched VDT

The build instructions for VDT give a simple one-line command to build the entire VDT distribution using GPT. As stated above, a specially patched version of VDT was used for EDG and LCG. Unfortunately, the changes made for these patches were

not reflected in the configuration files used by GPT to recurse through the source code directories when building the VDT distribution. Therefore, the documented GPT build command failed. In order to build VDT it was therefore necessary to build each sub-component separately.

4.3 Building VDT manually (with the assistance of GPT commands)

In fact, most of the sub-components of the EDG-patched version of VDT Globus built successfully using the GPT globus-build command specified with the appropriate set of configuration options. A few sub-components appeared to need slight variations to this, for example a supplied patch applying first but in general they all built successfully eventually. The time-consuming issues here were: -

- Determining the correct order in which to build sub-components as there are many inter-dependencies between sub-components.

- Determining the correct set of configuration options to specify for each sub-component

This stage involved a long sequence of trial-and-error experiments, coupled with searches through output and matching with build configuration files to locate the necessary option to set, or occasionally modification to be made to a configuration file.

The type of changes that were required to be made were: -

- Ensuring that that the correct compiler switches for IA64 were applied (namely fPIC and DPIC) by editing auto-tools configuration files.

- Fixing minor "bugs" in the packaging of the software, which resulted in the build failing. These appeared to be simply the result of incorrectly packaged source code (e.g. mismatched versions of vanilla openssl and VDT openssl patches bundled with EDG-patched VDT tarball). [6]

- Some components required special options of gpt commands to build them and this was undocumented meaning that these options had to be discovered by trial and error.

A Makefile that built the components in the required order was developed. This allowed individual components to be re-built easily with specified flavors if required, which proved very valuable whilst determining the dependency order and fixing problems when testing the installation.

4.4 External software required

No external RPMs were required to be re-built specially to allow the VDT build to complete.

---

[6] This was probably a consequence of the fact that the source distribution did not appear to be fully maintained as it was not an official distribution,

Myproxy was a required dependency for EDG and this is generally supplied as an additional VDT RPM in binary form or as an additional "bundle" in source form. It was therefore necessary to build a version of myproxy manually and to add this to the VDT build procedure so as to produce the required RPM.

A specially patched (for Globus) version of automake was required but this was distributed with VDT.

gcc version 2.96 as installed with Red Hat AS 2.1 for IA64 was used to compile and install VDT successfully. During the porting of EDG it transpired that a patched (for EDG) version of gcc 3.2.2 was required to build the EDG sources. This version was acquired and used in subsequent builds of VDT without any apparent problem but there was not a requirement for using this later version in order to build VDT.

4.5 Building VDT RPMs from the compiled and installed binaries

A set of scripts was developed for converting the compiled binaries into RPMs matching the IA32 versions distributed by the VDT team. These were adapted from an unsupported set of scripts that was acquired from a member of the VDT development team. This had several local environment specific features hard-coded into it but with some inspection of the scripts and trial and error, the necessary modifications to these scripts were made to produce a reproducible procedure.

4.6 Summary of issues encountered with porting VDT

VDT is generally distributed in binary form and the source code is not generally distributed (although it was available from the VDT web-site at the time this task was carried out). The main consequences of this for this task were: -

- The publicly available documentation for building VDT from source was limited and the documentation that existed was contained within various files within the distribution and was not up-to-date.[7]

- The documented build procedure using the Grid Packaging Toolkit (GPT) failed to work. This was a result of the fact that, at the time, the EDG software required a patched version of the standard distributed VDT software and the build files that GPT relied upon had not been modified to reflect these patches. It was therefore necessary to build the components of VDT step-by-step inferring the order of dependencies between components by source code inspection and trial and error.

As discussed, the procedure for distributing VDT has been re-organised and rationalised somewhat since this work was carried out. See chapter 8.1 for a discussion of ways that VDT binaries could be more easily obtained in future.

[7] It should be remembered that this was not part of the official part of the VDT distribution, and so full documentation would not necessarily be expected. This was, however a big factor in the effort involved for the task.

5. Producing IA64 binaries of EDG software

This chapter discusses the issues encountered in compiling an IA64 version of the EDG software components required to support the selected LCG grid node configuration from source code and then producing these in the RPM form desired to perform an LCG node installation.

5.1 Introduction

This part of the task involved building binary RPMs from the source code publicly available from the EDG project. RPMs for Work Packages 1, 2, 5, 6 and 7 of the EDG project were required.

In several cases, a "noarch" (i.e. platform independent) binary version of the RPM was required and so no re-compilation was necessary to produce these as re-use of the IA32 version was possible. It could still be considered whether it would be worthwhile to re-compile these from source on IA64 for optimisation reasons.

5.2 Overview of build system for EDG

Some features and characteristics that are worth noting about the EDG system are that it involves: -

- Large software development (about 200 persons working on the project)
- Long life cycle development (36 months)
- Distributed development of software (21 partners)
- Large scale integration (several Testbed sites)
- Software portability (Linux, Solaris, etc)

(From "DataGrid Quality Assurance" [5]). These issues therefore significantly increase the potential risk of complexity and inconsistency in source code management and build procedures across the project.

The document "Datagrid configuration management and build conventions" [3] was written within the EDG project (as part of WP12) and gives guidance to WP teams on using consistent naming conventions, project directories organisation and comments conventions, for all DataGrid middleware packages. (for Java, C++ and C code). It outlines conventions for CVS module structure, naming and tagging, as well as package organization.

5.3 Issues encountered during building of EDG RPMs

There were 2 basic categories of component that were encountered during the building of the required EDG RPMs: -

- Java/C++ components, built using the build tool *ant*.

- C/C++ components, built using *autotools*.

IA64 awareness had generally been built into the configuration files used by the build tools used and there were a very small number of porting issues with the source code, indicating that the code was generally "64-bit clean" (although as for with VDT, some compiler warnings were received regarding casting to integers of different sizes).

The main issues that needed to be addressed at this stage were: -

1. Understanding the build method that was required to be used and determining dependencies and build environment required for it. The build procedure was not clearly documented and details such as environment variable settings required were not clearly specified. It was necessary to inspect build files and attempt to infer the details of the process, along with trial and error.

2. Fixing/circumventing small problems encountered in the build process and fixing issues such as hard-coded Globus flavors[8] in the code/build files.

5.3.1 Compilers versions required

The following compilers were required to be used for building EDG software: -

- Java compiler from Java 2 SDK version 1.4.2_01

- gcc v3.2.2 (patched for EDG)[9]

Both these compilers had to be built from source as binary versions were not available at the required supported version.

6. Issues with acquiring IA64 RPMs for external software required

As stated, no external software was required for the building of VDT (other than the VDT additional "bundle" for myproxy). Some external RPMs were required in order to build the EDG RPMs.

A related issue here was that the supported platform for EDG was Red Hat version 7.3 whereas the installed operating system on IA64 machines was Red Hat AS v2.1 (and no version of version 7.3 for IA64 existed). In certain cases, dependency RPMs for the external RPMs were not at the required level by default, meaning that additional work had to be done to acquire/produce later versions of these RPMs.

The following basic categories of external RPMs that needed to be acquired for IA64 were encountered: -

---

[8] See section 4.2.2 above for a discussion of Globus flavors

[9] gcc 2.96 was previously the supported compiler for EDG but a phased migration to using gcc 3.2.2 for all EDG WPs was completed towards the end of 2003.

1. Unmodified external RPMs

2. Externals RPMs modified for EDG/LCG.

For many of these RPMs, it was simply a matter of locating a suitable RPM (and any required dependencies) on the web. For some external RPMs however, a suitable IA64 binary was not available. In this case, the first option was to try and locate a suitable source RPM (SRPM). In most cases, the SRPM built without problem. In a not insignificant number of cases, the SRPM build failed due to lack of IA64 support or some options incompatible with IA64. In this case, it was necessary to unpackage, modify and re-package the SRPM. Finally, in a small number of cases, it was necessary to build and install from source code without the use of RPMs as no suitable version of an RPM (binary or source) existed.

In some cases, this task was simplified by the fact that some repositories of external RPMs had been set up at EDG partner sites (although no complete repository containing all required external RPMs was discovered, even in relation to the officially supported autobuild system used within the project).

7. Summary of most important issues encountered

The majority of the work carried out centred around discovering the correct procedure and environment settings required for building sources. For both VDT and EDG, these were not fully documented. It was necessary to experiment with builds and interpret error and warning messages and examine source code in order to infer the build environment required for a successful build.

It should be stated that much more work was involved in producing the VDT RPMs than for the EDG RPMs. Factors in this were: -

- Once the correct build procedure was discovered, the EDG procedure worked OK. The published VDT build procedure had not been maintained and so failed, requiring significant reverse engineering to reproduce a working version.

- The EDG build procedure automatically produced RPMs from source with one build command. An additional procedure was required to convert VDT binaries to RPMs. A procedure for this was supplied by the VDT team on request but it required some local modification before it worked.

The reasons for these differences between the experience of building VDT and EDG are probably largely down to the fact that the EDG project had a clear policy of prescribing and enforcing the use of standardised build environments. Such a policy was essential given the nature of the EDG project with development spread across multiple teams (although a managed build policy should be part of any software development project anyway). In the case of VDT, development and distribution was carried out between closely working teams, which did not officially support a source code distribution, although they supplied an unsupported one, which was the one required by this project as well as by the LCG distribution team. This unsupported VDT source release would therefore not be expected to be under such tight quality

control measures. A supported version would however be highly desirable as is discussed in Chapter 8.

The issues that were encountered in the building of VDT and EDG could be dramatically reduced by taking the following steps: -

1. Clearly documenting the build procedure, including details such as required environment variable settings and RPM dependencies.

2. Routinely testing the IA64 build path so that elementary configuration errors were eliminated.

3. Making binary and source RPMs of external RPMs required for installing and building easily available from one repository (especially those that had been modified for EDG/LCG).

Very few additional issues were uncovered that actually related to porting of code to IA64. Apart from some rare instances where some long integer definitions had to be modified, changes were not necessary for IA64, although compiler warnings regarding integers of different sizes were often seen.

8. Suggestions for obtaining necessary binaries for IA64 in future

This chapter discusses some options that could be considered for obtaining IA64 binaries from now onwards.

8.1 VDT

Three main alternatives are available for producing updated versions of IA64 VDT binaries as new releases of VDT are produced: -

1. Request that the NMI /VDT teams produce and distribute an IA64 distribution of VDT.

2. Request that the NMI /VDT teams produce and distribute a version of the VDT source code that will build using the documented build procedure (and that the documentation and documented build procedure are kept up to date with source code changes).

3. Continue to maintain the method devised during this project.

Option 1 is the simplest and most efficient if the VDT distributors agree to undertake the extra work involved. This approach would leverage all the advantages of the NMI/VDT build centralised build procedures.

Option 2 is not very different from option 1 in terms of the work required, in that it is really only the place where the actual building is done that would change. Some advantages of this approach are that: -

1. It is more flexible in that individual sites can build for their particular machine types (and modify configuration options if the build procedure allows it simply).

2. It encourages partners to use and the source code and be involved in improving and maintaining it as has happened to date resulting in the assimilation of many patches from EDG, for example, into the official release of VDT (and eventually Globus)

3. It would be assumed that NMI/VDT would have done the work of producing the build procedure for their own internal needs so that it would just be a matter of releasing this.

Of course, some of the necessary patching and other work could be contributed by interested IA64 partners.

Option 3 requires the most work to be done by users of the software (such as OpenLab and LCG) and results in possible duplication of the same work across different sites. Given the reasonable options 1 and 2, it is probably not the preferred long-term solution although until successful agreement of one or both of those options, it is still necessary. This can result in significant effort, especially as the build methods and distributed software can change significantly between releases (as happened in March 2004).

The build procedure for VDT using GPT tools is in fact very simple (a single "gpt-build" command with suitable parameters) but requires that the GPT build and packaging information files are kept synchronised with the structure of the source code tree. In hindsight, given the effort involved with building VDT from source, it may have been worth the overhead of studying the GPT build mechanism and making the required changes to the GPT information files so as to support this simple method. This approach could be investigated if the distribution of an IA64-friendly version of VDT does not arrive.

8.2 EDG/LCG

A similar set of options exists as described above for VDT, namely: -

1. Request that the the development teams produce and distribute an IA64 distribution of the necessary (EDG?) RPMs for LCG

2. Ensure that the development teams produce and distribute reasonable documentation detailing the prescribed build procedure (the build procedures themselves were shown to work OK for IA64).

Of these options, number 2 would seem to be the preferred one as it would allow building from source and potential modification if required to allow development of IA64 patches, which could be contributed back into the distribution if agreed.

9. Conclusions

The majority of the work to support IA64 for VDT and EDG/LCG had already been done very well by the respective groups. However , two factors meant that this took a lot of time to convert to a working IA64 distribution: -

1. The lack of documentation of build procedures

2. The fact that the IA64 build path, although developed, had not been used and tested regularly,

The reasons for the lack of documentation are understood (no official source code distributions of VDT or EDG are supported at present). In hindsight, the actual problems encountered were relatively minor, once the cause of the problems was established. The development team's local undocumented knowledge would have been very valuable here.

None of the options discussed in chapter 8 would require significant changes to source or build files for VDT or EDG, although some additional work would need to be built into the respective distribution teams' procedures. The necessary source and build file changes could be contributed by an interested user group (such as LCG or OpenLab).

The ideal outcome of this could be that the respective VDT and EDG development teams routinely make available: -

1. A binary distribution for IA64, and

2. A source code distribution with documented build procedures

Of these, option 2 would be of the most benefit to groups such as the LCG distribution team as well as the Grid community as: -

- it allows modification and bug-fixing without going through an entire request and re-build cycle involving the respective development and distribution team

- it facilitates contributions of bug-fixes and enhancements from user groups such as LCG. This process (so far only possible due to large effort within the LCG team to develop and maintain procedures for re-building from source) has proved fruitful as many bug-fixes have been contributed to the VDT code base with many of these subsequently contributed to the main Globus distribution.

If a maintained source distribution of VDT and EDG for LCG software (with the IA64 build path routinely tested) was produced, it would not be strictly necessary distribute an IA64 binary version as this could be built from the sources, although it would be an additional service to those just wishing to install the latest supported version. A choice between options 1 and 2 might of course depend on other (political) factors as well as the technical aspects.

References

[1] "European DataGrid project" - http://eu-datagrid.web.cern.ch/eu-datagrid/

[2] The Virtual Data Toolkit (VDT) - http://www.lsc-group.phys.uwm.edu/vdt/

[3] "Datagrid configuration management and build conventions": Yannick Patois, CNRS, IN2P3. Available at http://datagrid.in2p3.fr/d6.2/DataGrid-ConfMngmt-BuildConv.html

[4] "DataGrid 3rd EU review. Available at http://agenda.cern.ch/fullAgenda.php?ida=a036278

[5] "DataGrid Quality Assurance", Gabriel Zuquine , CERN. Available at http://eu-datagrid.web.cern.ch/eu-datagrid/1Y-EU-Review-Material/cd-1y-eu-review/1-datagrid-1y-eu-review-agenda&presentations/agenda/..%5CPresentations%5C6-Quality%20Assurance%5CEU-1Y-Review-Quav1r2%5B1%5D.pdf