Integration, Deployment and Scientist Interface

LASER Summer School on Software Engineering 2013, Elba Island, Italy
Pere Mato / CERN
The Main Challenge

- A scientist job ‘lands’ in an unspecified computer node of an unspecified computer center somewhere in the Grid/Cloud
  - First, the scientist job will need the experiment libraries installed in the computer node
    - With the right version of the software (with all dependencies) for the adequate platform (OS version, set of system libraries and services)
    - At the same time other scientists may require other versions of the same software with very distinct set of dependencies
  - Second, the job will need to access the input data using some sort of logical name independent of the actual location
  - Third, the results will need to be publish/uploaded such that can be recovered elsewhere

How to handle the first challenge?
Reminder: The Software Stack

- This is what the “job” needs in terms of software “installed” in the Grid/Cloud node
  - Several hundreds of shared libraries properly configured, for the right version and platform (OS+compiler)
  - A proper environment setup
  - Minimal requirements in terms of OS services and libraries
- Other jobs from other scientists from probably other experiments may be running in the same node using a different set of libraries and versions
Two Possible Approaches

* Install/deploy all versions of the software in all computer nodes
  * Not an easy task considering that
    * there are $O(100k)$ nodes
    * new versions are released every week
    * keeping several versions may require incompatible underlying versions
    * jobs using a new version can only start once its software has been installed
    * etc.

* Make use of the virtualization and deploy/instantiate VM images containing everything needed
  * Probably better but still difficulties in
    * managing large VM images (creating and moving them)
    * new images will need to be made for every software release
    * security issues related to trusting the images
Horizontal Integration

- Traditional model
  - Horizontal layers
  - Independently developed
  - Maintained by the different groups
  - Different lifecycle

- Application is deployed on top of the stack
  - Breaks if any layer changes
  - Needs to be certified every time when something changes
  - Results in deployment and support nightmare
Vertical Integration

- Application driven approach
  - Analyzing application requirements and dependencies
  - Adding required tools and libraries
  - Building minimal OS
  - Bundling all this into Virtual Machine image

- Virtual Machine images should be versioned just like the applications
  - Assuring accountability to mitigate possible negative aspects of newly acquired application freedom
Vertical Integration

- Application driven approach
  - Analyzing application requirements and dependencies
  - Adding required tools and libraries
  - Building minimal OS
  - Bundling all this into Virtual Machine image

- Virtual Machine images should be versioned just like the applications
  - Assuring accountability to mitigate possible negative aspects of newly acquired application freedom

Diagram:

- Application
- Libraries
- Tools
- Databases
- OS
Virtual Machine Integration

- Application driven approach
  - Analyzing application requirements and dependencies
  - Adding required tools and libraries
  - Building minimal OS
  - Bundling all this into Virtual Machine image

- Virtual Machine images should be versioned just like the applications
  - Assuring accountability to mitigate possible negative aspects of newly acquired application freedom
Rethinking Application Deployment

- Emphasis in the ‘Application’
  - The application dictates the platform and not the contrary
- Application (e.g. simulation) is bundled with its libraries, services and bits of OS
  - Self-contained, self-describing, deployment ready
- What makes the Application ready to run in any target execution environment?
  - e.g. Traditional, Grid, Cloud

Virtualization is clearly the enabling technology here
Main Difficulties with Virtualization

- **Performance**
  - Fear that the performance will much lower than bare-metal
- **Management tools and standard interfaces**
  - Many open source and commercial solutions
  - EC2 becoming de-facto standard API
- **Managing large VM images**
  - Distributing large images to many centers can be a problem
- **Trusting VM images**
  - Many sites do not trust user provided images
- **Contextualization**
  - Need to customize images to specific function
- **Adequate storage architecture for HEP**
  - Large data access requirements
- **Costs**
  - Cost comparisons between public and private clouds
It seemed that Virtualization could provide the solution we were looking at

**CernVM** is a R&D project started 4 years ago on Virtualization at CERN

The **CernVM** image is an attempt to mitigate the mentioned difficulties (performance, image distribution, trust, contextualization, etc.)

* Tuned for best performance of HEP applications
* Single image fits all [LHC] experiments
* Very small is size (only 250MB) with just-enough OS (much better with the new μCernVM development)
* Experiment software is factorized out (dedicated File System)
* Flexible configuration and contextualization mechanisms

http://cernvm.cern.ch
CernVM Elements

* Minimal Linux OS (Scientific Linux)
* CernVM-FS - HTTP network file system optimized for just-in-time delivery of experiment software
* Flexible configuration and contextualization mechanism based on public Cloud API
Initial Scope

- Checkout
- Compile
- Test
- Debug
- Submit to Grid
- Display results
- Suspend
- Resume
CernVM comes with the read-only file system (CernVM-FS) optimized for software distribution

Main features:

- Very little fraction of the experiment software is actually used (~10%)
- Very aggressive local caching, web proxy cache (squids)
- Transparent file compression
- Integrity checks using checksums, signed file catalog, etc.
- Operational in off-line mode

No need to install any experiment’s software in each node

- ‘Virtually’ all versions of all applications are already installed
- The user just needs to start using it to trigger the download
Software File Size

50 % files are less than 4kB
Only 10% files required at runtime

cf. Tanenbaum et al. 2006 for “Unix” and “Webserver”
Comparison CernVM-FS / NFS II

Many tests has been run comparing CernVM-FS with NFS, AFS, etc.

- Better scalability with number of concurrent jobs
- Very modest load in the server: ~5 MB/s, 20 requests per second on CERN Replica
The Client: A Read-Only File System in User Space
The Server: A Publish Interface using Union-FS

- Kernel-level **Union File System** (AUFS)
- < 5% performance loss (untar)

- Fully POSIX-compliant read-write file system
- Encapsulated change set in scratch area
- In contrast to file-wise write: publishing of new snapshot
Between consecutive software versions: only ~15% new files
Software vs. Data

Based on ATLAS Figures 2012

<table>
<thead>
<tr>
<th>Software</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSIX Interface</td>
<td>put, get, seek, streaming</td>
</tr>
<tr>
<td>File dependencies</td>
<td>Independent files</td>
</tr>
<tr>
<td>$10^7$ objects</td>
<td>$10^8$ objects</td>
</tr>
<tr>
<td>$10^{12}$ B volume</td>
<td>$10^{16}$ B volume</td>
</tr>
<tr>
<td>Whole files</td>
<td>File chunks</td>
</tr>
<tr>
<td>Low latency</td>
<td>High throughput</td>
</tr>
<tr>
<td>Absolute paths</td>
<td>Any mount point</td>
</tr>
<tr>
<td>Open source</td>
<td>Confidential</td>
</tr>
<tr>
<td>WORM (&quot;write-once-read-many&quot;)</td>
<td>Versioned</td>
</tr>
</tbody>
</table>
Content Distribution

* Stratum Model
  * + Fast and Scalable
  * + No single point of failure
  * - Complex hierarchy
Client-Side Fail-Over

- **Proxies**
  - SL5 Squid, load-balancing + fail-over
    - e.g. `CVMFS_HTTP_PROXY="A | B | C"`

- **Mirrors**
  - Fail-over mirrors at CERN, RAL, BNL
    - For roaming users automatic ordering based on RTT
Client-Side Fail-Over

* Proxies
  * SL5 Squid, load-balancing + fail-over
e. g. `CVMFS_HTTP_PROXY="A | B | C"`

* Mirrors
  * Fail-over mirrors at CERN, RAL, BNL
  For roaming users automatic ordering based on RTT
Each experiment is given a VM to install and test their software using own installation tools

Publishing is an atomic operation
Turn-Around for Software Distribution

Improvement: from days (Grid Installation Jobs) to hours
Best solution so far

- The CernVM-FS provides the best solution to LHC
  - CernVM-FS proved to be scalable and reliable in production
  - All WLCG sites depend on CernVM-FS for software distribution by 2013
  - Upcoming features mainly focus on performance optimization and stabilization
  - All experiments has adopted it as the channel to distribute the software to all the Grid and Cloud resources

- Software installation done centrally and only once
  - Saving a lot of resources and headaches

- From the scientist view point, it really solves the issue of having all software all the time available everywhere
μCernVM

• Idea: Operating system itself on CernVM-FS
  • Instead of 400 MB hard disk image: 10 MB ISO image + 100 MB cache.
• Not a LiveCD, not a diskless node

Operating System on Demand!!
μCernVM changes the VM Life Cycle

1. Plan
2. Prepare Repositories
3. Build
4. Test
5. Endorse

Development Cycle

10. Feedback
9. Terminate
6. Instantiate
7. Contextualize

Deployment Cycle

8. Monitor
11. Retire

CernVM Infrastructure

User Infrastructure
μCernVM changes the VM Life Cycle

CernVM Infrastructure

User Infrastructure

1. Plan
2. Prepare Repositories
3. Build
4. Test
5. Endorse
6. Instantiate
7. Contextualize
8. Monitor
9. Terminate
10. Feedback
11. Retire
μCernVM changes the VM Life Cycle

Avoids image building
Solved image distribution
References

- Decentralized Data Storage and Processing in the Context of the LHC Experiments at CERN

- CernVM-FS: Delivering Scientific Software to Globally Distributed Computing Resources
  J. Blomer, P. Buncic, and T. Fuhrmann
  Proc. of the 1st Workshop on Network-Aware Data Management held in conjunction with the IEEE/ACM International Conference for High Performance Computing, Networking, Storage and Analysis (SC’11), Seattle, 2011

- A practical approach to virtualization in HEP,
  P. Buncic, C. Aguado Sánchez, J. Blomer, A. Harutyunyan, and M. Mudrinic
  The European Physical Journal Plus, 126(1), 2011

- Distributing LHC Application Software and Conditions Databases using the CernVM File System
  J. Blomer, C. Aguado Sánchez, P. Buncic, and A. Harutyunyan

- A Fully Decentralized File System Cache for the CernVM-FS
  J. Blomer and T. Fuhrmann
  Proc. 10th Computer and Communications Networks (ICCCN’10), Zürich, 2010

- LHC Cloud Computing with CernVM
  B. Segal, P. Buncic, D. Garcia Quintas, C. Aguado Sánchez, J. Blomer, et al.
  Proceedings of Science, ACAT(004), 2010
Setting the Scene

- We have the **experiment software available** (full stack) ready to be used in all the nodes of the Grid/Cloud
- We have also available the processed data (reconstructed, pre-selected) **distributed among the different computer centers**
- **Data Analysis** is the full data processing chain from reconstructed event data up to producing the final plots for publication (iterative process)
  - Reduce data samples to more interesting subsets (selection)
  - Compute higher level information, redo some reconstruction, etc
  - Calculate statistical entities
- **Algorithm development** is essential in data analysis
  - The ingeniousness of the scientist is materialized in code (C++, Python,...)
- The need to use a **very large computing facility** distributed worldwide
  - They will need to understand it and use it correctly
Some Obvious Facts

- The large amount of data (~15 PB/year) to be analyzed and the computing requirements prevents the idea of non-distributed data analysis
- The scale of ‘distribution’ is a **continuum** and goes from a local cluster to a computer center or to the whole federation of grid(s) or cloud(s)
- **Distributed analysis complicates the life of the physicists**
  - The old times with a large ‘mainframe’ made data analysis simpler
  - In addition to the actual data analysis code, he/she has to worry about many other technical issues
LHC Analysis Data Flow

- Data is generated at the experiment, process and distributed worldwide (T1, T2, T3)
  - See introduction lecture
- The data analysis will process, reduce, transform and select parts of the data iteratively until it can fit in a single computer
  - The final goal for the physicist is to be able to perform the final data analysis and presentation in his/her personal computer
- How this is realized?
HEPCAL-II† Dreams

- All elements there and still valid
  - Less organized activity (chaotic)
  - Input data defined by asking questions
  - Data scattered all over the world
  - Own algorithms
  - Data provenance
  - Software version management
  - Resource estimation
  - Interactivity

- Advocating for a sophisticated WMS (workload management system)
  - Common to all VO’s (Virtual Organizations)
  - Plugins to VO’s specific tools/services

† Use cases for a HEP Common Application Layer for Analysis, LCG-2003
HEPCAL-II Reality

- The implementation has evolved into a number of VO specific “middleware” using a small set of basic services
  - E.g. DIRAC, PanDA, AliEn, Glide-In
- Development of ‘user-friendly’ and ‘intelligent’ interfaces to hide the complexity
  - E.g. Crab, Ganga
- Not optimal for small VOs that cannot afford to develop specific services / interface
  - Or individuals with special needs
Analysis Software

* Specialization of the Experiment’s Frameworks and Data Models for data analysis to process reconstructed and analysis objects
  * CMS Physics Analysis Toolkit (PAT), ATLAS Analysis Framework, LHCb DaVinci/LoKi/Bender, ALICE Analysis Framework
  * In same cases selecting subset of Framework libraries
  * **Collaboration approved** analysis algorithms and tools

* Other [scripting] languages have a role here
  * PYTHON is very popular in addition to the C++ macros of ROOT
  * Ideal for prototyping new ideas

* The Physicist typically develops its own Algorithm(s) based on these frameworks but also is willing to replace parts of the official release
Front-End Tools

Ganga
- GUI
- CLI
- script

Ganga Core
- Application Plugins
  - Executable
  - Gaudi LHCb
  - Athena LHCb
- Persistence Manager
  - Job Repository
  - Metadata
- Job Plugins
  - Localhost
  - EGEE Grid
  - NorduGrid
  - Condor
  - DIRAC LHCb
  - ssh
  - PANDA LHCb

ALICE
- PROOF SETUP
- MY MACHINE
- ALIEn SETUP

Crab
- Job Submission
- Task Assignment
- Monitoring
- Resource Management
- Task Registration
- Dataset Management
- Job Submission

CERN

Tuesday, September 10, 13
Major Differences

- Both **Ganga** and **ALICE** provide an interactive shell to configure and automate analysis jobs (Python, CINT)
  - In addition **Ganga** provides a GUI

- **Crab** has a thin client. Most of the work (automation, recovery, monitoring, etc) is done in a server
  - This functionality is delegated to the VO specific WMS for the other cases

- **Ganga** offers a convenient overview of all user jobs (job repository) enabling automation

- Both **Crab** and **Ganga** are able to pack local user libraries and environment automatically making use of the configuration tool knowledge

- For **ALICE** the user provides .par files with the sources
Analysis Activity

- [1] Algorithm development and testing starts locally and small
  - Single computer → small cluster

- [2] Grows to a large data and computation task
  - Large cluster → the Grid/Cloud

- [3] Final analysis is again more local and small
  - Small cluster → single computer

- Ideally the analysis activity should be a continuum in terms of tools, software frameworks, models, etc.
  - LHC experiments are starting to offer this to their physicists
  - **Ganga** is a good example. From inside the same session you can do a large data job and do final analysis with the results
Input Data

- The user specifies on what data to run the analysis using VO specific dataset catalogs
  - Specification is based on a query
  - The front-end interfaces provide functionality to facilitate the catalog queries
  - Each experiment has developed event tags mechanisms for sparse input data selection

- Data scattered over the world
  - Computing model and policies of the experiment dictate the placement of data
  - Read-only data with several replicas
  - Portions of the data copied to local clusters (CAF, T3, etc) for local access
Output Data

- Small output data files such like histogram files are returned to the client session (using the sandbox mechanism)
  - Usually limited to few MB

- Large output files are typically put in Storage Elements (e.g. Castor) and registered in the grid file catalogue (e.g. LFC) and can be used as input for other Grid jobs (iterative process)

- Tools such as CRAB and Ganga (ATLAS) provides strong links with VO’s Distributed Data Management/Transfer systems (eg. DQ2, PhEDEx) to place output where user wants it
Submission Transparency

* The goal is to make it easy for physicists
* Distributed analysis as simple as doing it locally
  * Which is already complicated enough!!
  * Hiding the technical details is a must
* In **Ganga** changing the back-end from LSF to DIRAC requires to change one parameter
* In **ALICE** changing from PROOF to AliEn requires to change one name and provide a AliEn plugin configuration
* In **CRAB** changing from local batch to gLite requires a single parameter change in the configuration file
The Ganga Mantra

Configure once, run anywhere!!

- Test locally and with small sets of jobs before submitting 10k!
- This solves 99% of your “Grid” problems
Ganga Features

• GANGA handles the complete life cycle of a job:
  • Build → Configure → Split → Submit → Monitor → Merge

• GANGA does the following (and much more) for the user:
  • builds/compiles applications;
  • configures jobs, including building input sandboxes, to run on user-specified backends;
  • submits jobs locally, to batch systems and to the grid;
  • monitors jobs and updates the user on any status changes;
  • automatically retrieves output when jobs complete;
  • merges output (if requested)
Some more Ganga Details

- GANGA is written in Python and has an enhanced Python prompt (IPython) that supports:
  - Python syntax
  - Shell commands
  - TAB completion, scrolling through your history, etc.

- It’s similar to working on the command line except Python syntax is valid and TAB completion works for Python objects, methods, variables, etc.

- GANGA is “Job” oriented
Example Ganga Job

* To run DaVinci (LHCb Analysis application) tutorial, in GANAGA simply do:

```python
In[1]: j = Job()
In[2]: j.application = DaVinci(version='v26r3p2')
In[3]: j.application.optsfile = ['<path>/DaVinciTutorial.py','<path>/Bs2JPsiPhi.py']
In[4]: j.backend = Interactive()
In[5]: j.outputsandbox = ['DVHistos 1.root']
In[6]: j.submit()
```

* To run on the Grid, simply do `j.backend = Dirac()`

* GANAGA will automatically collect all of your modified files and send them with the job
Split and Merge

- Trivial form of parallelism
- In HEP we have the huge advantage that each particle collision is independent of each other
  - So, we can process in parallel several sets of collisions (files with events) and merge the results (statistical quantities) afterwards
- The Scientists Interface often provides facilities to split the user job into a set of sub-jobs that can be submitted in parallel and facilities to merge the results
  - Sub-jobs are distributed and run where the data is or where can be easily obtained
  - The result merging may imply to provide a some user code (plugin) to know what needs to be merged
Example: Ganga

* Tools like Ganga provides a rich set of *splitters* and *mergers*
  * Splitting a job into sub-jobs by a number of criteria (files, input arguments, etc.)
  * Merging the results (file concatenation, summing histograms, etc.)
* Monitoring and bookkeeping tools maintains the job/sub-job relationship

```
In [1]: j = Job()
In [2]: j.application = Root(script='test.C')
In [3]: j.backend = Dirac(settings={'CPUTime':60})
In [4]: j.splitter = ArgSplitter(args=[['a'],['b'],['c']])
In [5]: j.merger = TextMerger(files=['test.txt'])
In [6]: j.outputsandbox = ['test.txt']
In [7]: j.submit()
```
Example: Processing TTree

- ROOT PROOF facility exploits this “trivial form of parallelism”
- The User “only” needs to provide a TSelector class
- The system takes care of packetizing the data and executing the TSelector:Process()
- The results are collected and merged automatically
Take Away Messages

- Distribution and installation of the LHC software stack is highly non-trivial
  - quite large, many concurrent versions, very dynamic, etc.
  - It has required several iterations to finally converge into a simple, yet powerful, content delivery network optimal for software
- Once the software and the data are available at the distributed computing system (Grid/Cloud) we need to assist the scientists to ‘hide’ as much as possible its complexity
  - Experiments has developed a variety of tools and systems to assist scientists to submit their data analysis “jobs” to the Grid/Cloud and collect the results
  - Several back-ends (workload management systems) take care of distributing the jobs according to available resources, data availability, additional rules, etc.