

**Imperial College
London**

**LHC*b* status and plans for
distributed analysis**

7 March 2005

What is analysis in LHC*b*?

The aim of LHC*b* is to extract results on *CP* violation from rare *B*-meson decays.

Detector will see the full event rate of 40 MHz from LHC collisions.

When data leaves the detector the rate will be around 2 kHz (200 Hz physics, 1800 Hz systematics and calibration).

Output data in DST format will be divided into streams each containing maybe 10^7 events per year.

These are the events that the end user will access for analysis.

Size for DST that user will see is about 100 kB per event (for data).

For analysis we also need access to:

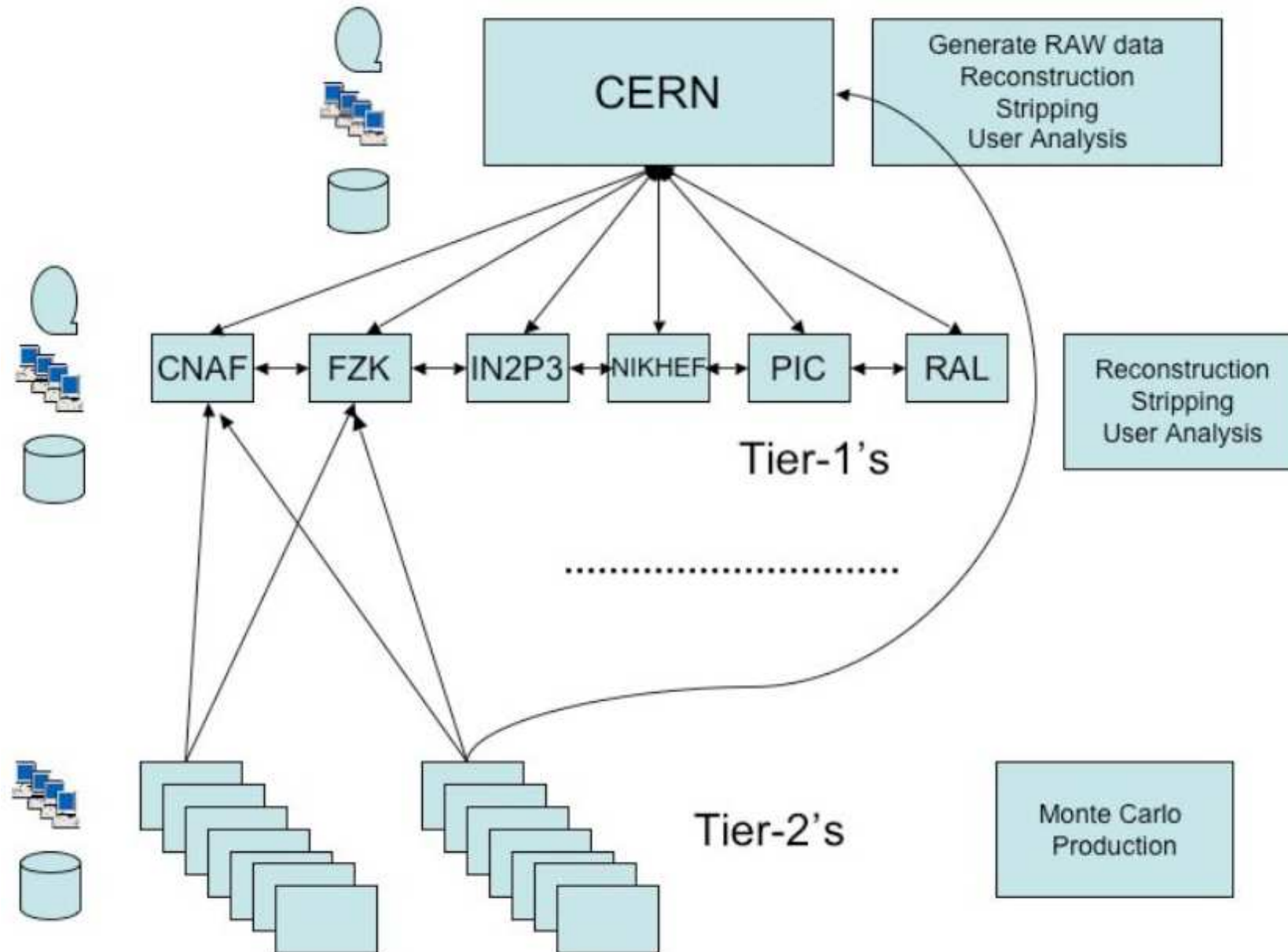
Simulated data for signal samples and inclusive events.

2 times larger size.

Systematic data sample

Will be stripped 4 times a year.

LHCb computing model with 2 kHz HLT rate



Use of Tier centres

CERN:

Reconstruction of *B* sample, analysis, stripping.

Tier 1:

Reconstruction, stripping and analysis.

Tier 2:

Mainly MC production.

	CERN	Tier1's	Tier2's	Total
Stripping	0.17	1.03	0.0	1.20
Full reconstruction	0.40	2.42	0.0	2.82
Monte Carlo	0.0	0.0	7.6	7.6
Analysis	0.32	0.97	0.0	1.29
Total	0.90	4.42	7.65	12.97

Usage in MSI2k years for 2008.

The Ganga project

The purpose of Ganga is to work as a wizard for LHC*b* users running Gaudi applications.

Mainly we have running C++ analysis applications (DaVinci) in mind.

We need to support the following behaviour:

writing new Gaudi algorithms, either in C++ or Python;

modifying existing algorithms;

modifying job options of existing/new algorithms.

Data will now (and in the future?) be distributed mainly at Tier 1 centres but we should support Tier 2 and local data as well.

The input data will be data in POOL format.

Output from distributed analysis will be a combination of data in POOL format, ROOT/HBOOK files and stdout/stderr.

A quick reminder of Ganga

Jobs in Ganga move between different states.

This gives the bookkeeping

The user can monitor jobs in the states:

New

Configured

Submitted

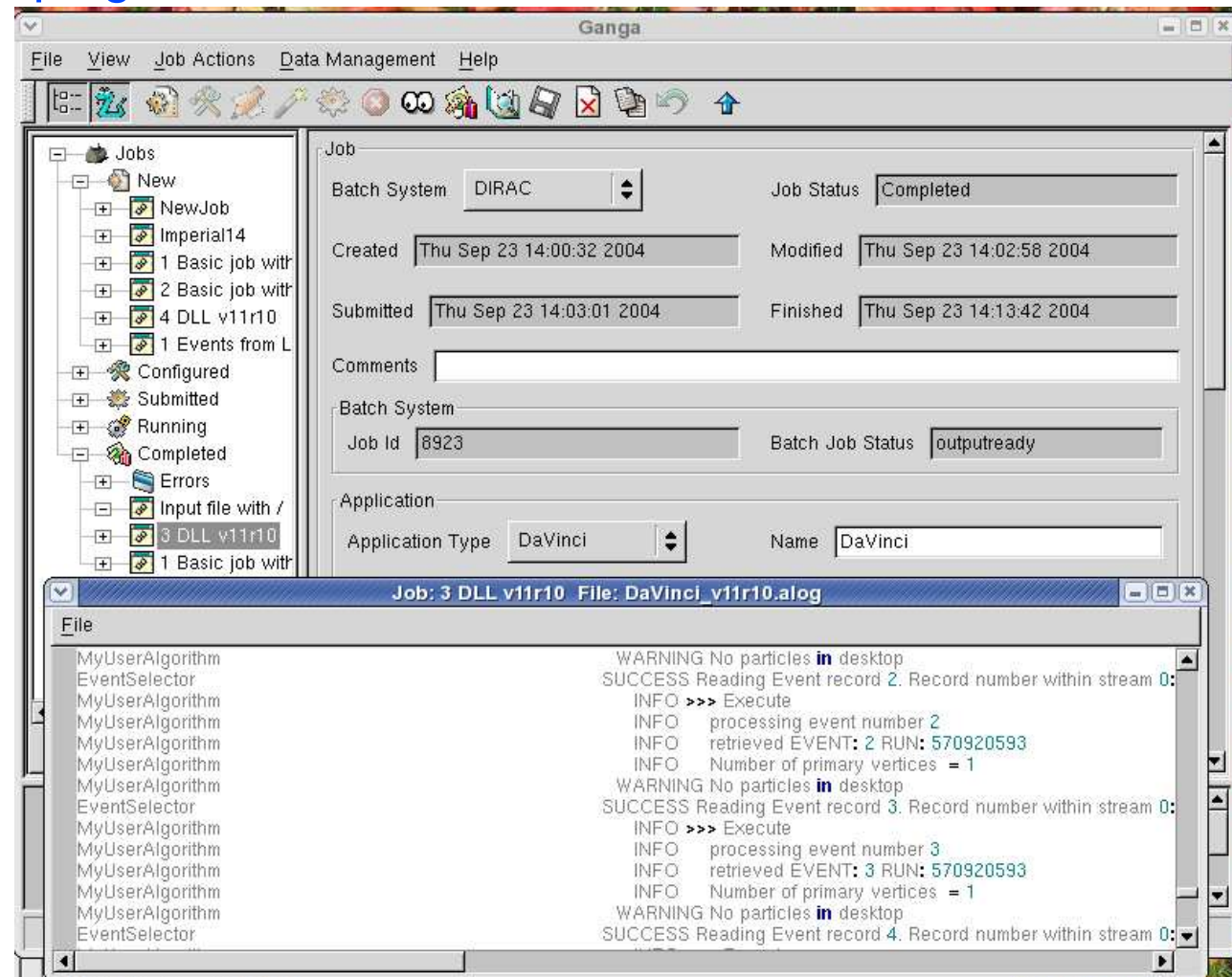
Running

Completed

Error – if reported!

Jobs are preserved between invocations.

Status updated at regular intervals.



CLIP - Command Line Interface in Python

A GUI is good for providing overview and for new users to learn about distributed analysis.

Many advanced users would like to write scripts that perform repeated operations.

Solution is to have a python based CLI that interoperates with the GUI.

A syntax has been defined and interfaced with the existing framework.

Many additional benefits

- Will allow test jobs to be defined and run in new releases in an automatic way.

- Will allow prototyping of new plugins for applications or submission systems to be tested without initially worrying about GUI.

- Will allow automatic tests of new LHC*b* releases to be defined.

Will illustrate the cycle of an LHC*b* analysis in terms of CLIP.

Ganga releases

Current release is Ganga 3.0

Since last public release in October, many new features.

CLIP.

More features for LHC*b* application handler.

Update of Dirac submission handler.

Implementation of ATLAS Athena application handler.

Support of binary dependencies on SLC3.

Documentation completely updated.

This will be last major release in current framework.

Bug fixes and minor enhancements will be made.

Ganga 4

Now focus is on refactored framework.

Andrew to talk about this.

Trivial example of using CLIP

Create a job:

```
dv = DaVinci()
dv.optionsfile='myoptions.opts'
# create a new job
j = Job(name='myjob', application=dv)
print 'created job ',j.id
j.submit()
```

List jobs:

```
>>> print jobs
Statistics: 11 jobs jobs
-----
      ID      status      name
#   1  completed  Hello World
#   2  completed  Copy of LSF
#   3  completed           LSF
#   4  completed      CMTUSER
#   5  completed           Root
#   6  completed      Loop1
```

A user view of an analysis – create algorithm

An analysis will define a work flow of algorithms to run.

The algorithms will be written mainly in C++ and run inside the Gaudi framework.

Possibility to write algorithms in Python as well but no widespread use (yet)

Code for analysis will be a mixture of

Standard LHC*b* algorithms

Under version control

User modified standard algorithms

This is typically for development of the standard tools

User specific algorithms

These might be different from job to job.

No version control.

User supply of DLLs present major difference to production jobs.

A user view of an analysis – configure job

LHC**b** jobs are configured through a set of options files.

These files have no state and can be pre-processed in a static way.

This is an important point for distributed analysis model.

For analysis these options might be different from one job to the next.

Example:

```
ApplicationMgr.TopAlg += { "PreLoadParticles" };

// Set to use the CombinedParticleMaker
PreLoadParticles.PhysDesktop.ParticleMakerType =
"CombinedParticleMaker";

// Default values for particle types to be made
// Note that when exclusive mode is selected the selection is
done in
// the order set below
PreLoadParticles.PhysDesktop.CombinedParticleMaker.Particles =
{"kaon", "pion" };
```

Create an analysis job

CLIP code:

```
# define DaVinci application
dv = DaVinci()

# tell we use a non-default location of code
dv.cmt_user_path='/afs/cern.ch/user/u/uegede/cmttest'

# take options file from user area
dv.optionsfile=dv.cmt_user_path + \
    '/Phys/FlavourTagging/v5r4/options/DVBTagging.opts'

# create a job
j = Job(application=dv)
```

A user view of an analysis – test and run algorithm

For testing and running analysis the average physicist will always do what seems the easiest way to get a result tomorrow!

Interaction with complicated systems that might bring long term time savings never gain wide acceptance.

```
CLIP code: # create a new job that go to batch
            j = Job(application=dv, backend='LSF')

            # change queue and submit
            j.queue='1nh'
            j.submit()

            # make copy and send to Dirac
            j2 = j.copy()
            j2.backend='Dirac'
            j2.submit()

            # from user point of view jobs are treated in the same way
            print j,j2
```

A user view of an analysis – test and run algorithm

For running larger scale jobs the incentive will always be to use what worked yesterday.

For larger scale analysis jobs this means the LSF batch system at CERN.

To change the behaviour to be for GRID jobs we can.

Make it easier to use than LSF

Tough, but our aim should be that it at least is not harder.

Limit the available CPU and data resources at CERN.

Seems to happen at the moment by default...

Ganga, as in example on last page, try to make all types of running equivalent.

A user view of an analysis – define dataset

Dataset for analysis needs specification.

Selection can happen in many ways

Fixed list taken from static web page.

Selection in a database

“The B to D_sK background calibration set”.

“Same data as I used yesterday”.

Specific event

Event 2134539 from run 3473.

“My usual test data”

Notice that none of these cases contain anything about LFN, PFN or other file specific information.

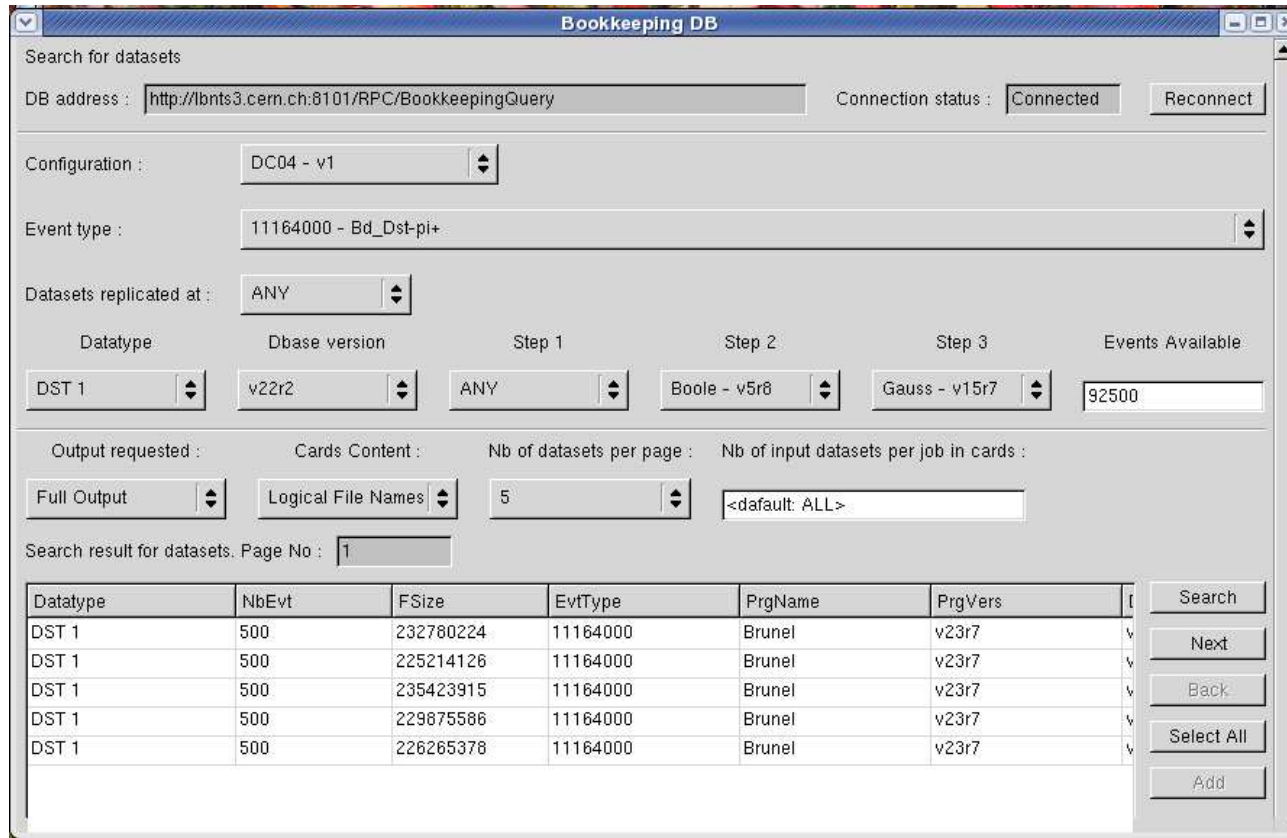
We want to introduce the concept of a *saved dataset* in Ganga

This should cover all the above use cases.

Only for Ganga-4.

Interaction with LHCb bookkeeping database

This will allow user to pick the data for analysis directly from within Ganga.



The screenshot shows a web interface for the Bookkeeping DB. The title bar reads "Bookkeeping DB". The interface includes a search bar, a DB address field with the value "http://lbnts3.cern.ch:8101/RPC/BookkeepingQuery", and a "Connection status" indicator showing "Connected" with a "Reconnect" button. Below this are several configuration fields: "Configuration" set to "DC04 - v1", "Event type" set to "11164000 - Bd_Dst-pi+", and "Datasets replicated at" set to "ANY". A table of search criteria is visible with columns: Datatype (DST 1), Dbase version (v22r2), Step 1 (ANY), Step 2 (Boole - v5r8), Step 3 (Gauss - v15r7), and Events Available (92500). Below the table are fields for "Output requested" (Full Output), "Cards Content" (Logical File Names), "Nb of datasets per page" (5), and "Nb of input datasets per job in cards" (<default: ALL>). A "Search result for datasets. Page No: 1" label is present above a table with 5 rows of results. The table has columns: Datatype, NbEvt, FSize, EvtType, PrgName, and PrgVers. To the right of the table are buttons for "Search", "Next", "Back", "Select All", and "Add".

Datatype	NbEvt	FSize	EvtType	PrgName	PrgVers
DST 1	500	232780224	11164000	Brunel	v23r7
DST 1	500	225214126	11164000	Brunel	v23r7
DST 1	500	235423915	11164000	Brunel	v23r7
DST 1	500	229875586	11164000	Brunel	v23r7
DST 1	500	226265378	11164000	Brunel	v23r7

Not clear how if we need a CLI for finding datasets.

A user view of an analysis – keeping track

For production of simulated events we know the importance of very good bookkeeping.

In an analysis situation users love it if it comes for free.

The default way of working is to sort out in retrospect which jobs failed, which produced corrupt output etc.

```
p = re.compile('^DsKanalysis.*')
n = m = 0

for j in jobs:
    if p.match(j.name):
        m = m+1
    if j.status()=='completed': m = m+1

print n, '/', m, 'jobs completed so far.'
```

A user view of an analysis – merging

The last stage of an analysis is to pull all the results together.

Merge all histogram files.

Chain ROOT output files.

Normalise to analysed
luminosity.

Search stdout for specific
patterns.

Not nice but seen a lot.

```
#Extract number of accepted events from stdout of
# jobs 'Loop1' – 'Loop7'
jobname='Loop'
p = re.compile('^'+jobname+'.*')
accept = re.compile('.*of events (processed|accepted).*')
files=[]
for j in jobs:
    if p.match(j.name):
        if j.status == 'completed':
            fname = j.directory+'/output/std.out'
            outfile = file(fname)
            files.append(fname)
            for l in outfile.readlines():
                if accept.match(l): print l
            outfile.close()
```

Testing merging with Root

As all in Python we can use pyRoot to merge and analyse ROOT output.

Proof of concept made.

```
# Interoperation between Ganga and ROOT
from ROOT import gROOT,TFile,TH1
```

```
for j in jobs:
    if j.status == 'completed':
        f = TFile(j.directory+'/output/test.root')
        h=f.FindObjectAny('2')
        h.Draw()
        raw_input("Press Enter To Continue: ")
        break
```

User requirements for distributed analysis system

For all users:

Responsive

Robust

Reliable

In addition for new users

Intuitive.

Clear error reporting.

Transparent to changes in middleware.

For more powerful users

Ability to deal with Event data collections.

Scripting language for job control.

LHCb requirements

Ability to set priorities

Within LHCb we need the ability to control how resources allocated to LHCb are used.

Accounting

From our current production we have seen the efficiency of monitoring to track down errors and wrong configurations in hardware/LCG/jobs.

Monitoring at the job level required of:

- CPU usage

- Memory usage

- Data requests

- Efficiency

- Users

Getting the analysis job to the Grid

Our aim is to get LHC*b* analysis to run on LCG resources.

Our adopted solution is to use infrastructure of production system (DIRAC) as an intermediate step.

We get most monitoring for free.

Only trivial DIRAC UI to install (one click).

We can use the experience of running production jobs on LCG.

We get access to all kind of sites (so not only LCG) running Dirac agents.

Not only simple use cases

Several personal development areas supported.

Release area can be changed on a *job-by-job* basis as well.

Some ideas on setting up automatic tests of new LHC*b* general s/w releases.

- Create a set of standard jobs.

- Monitor execution time and memory usage.

- Analyse output.

Developing new ideas

How to get towards a merging capability

Write use cases

Pre job merging

A user defines before a job is running how the output data from the sub jobs should be merged. When all parts of the job have finished the ones that terminated successfully have their output merged without any user interaction.

Post job merging

After the jobs have finished the user selects a merging strategy from a predefined list. All successfully finished parts of a job are included in the merging.

Get part of use case working

You saw example earlier on merging stdout.

Abstract behaviour to get syntax and API.

Review pseudo code

Test out new module

Status of analysis part of Data Challenge '04

Aim is to demonstrate that any physicist within LHC*b* can run an analysis on distributed resources.

Required steps:

Develop GAUDI algorithm for data analysis:

Debug and test on small dataset on local disk.

Working and demonstrated.

Define dataset to analyse:

Selection through bookkeeping system.

Working and demonstrated.

Divide dataset into bits that fit individual analysis jobs.

Not yet automatic but not a show stopper.

Status of analysis part of Data Challenge '04

Requires steps (cont):

Submit analysis

No specific knowledge of data location required.

Working; is part of data selection.

No need to specify location where job will run.

Implemented but not tested in anger.

Retrieve results

Get the output from jobs back to a single point

Working and tested.

Merge output data.

Not yet automatic. Not a big issue.

So what is the status of putting it all together?

Unfortunately much behind original schedule and not started yet.

Insufficient manpower main reason

Status of GANGA for LHCb analysis from Oct '04

Submission	Local	Working
	LSF	Working
	DIRAC	Working
	LCG proxy transfer	In Progress
Job Options	Selection	Working
	Editing	Working
	Expansion	Working
	Logical view	Deferred
Job handlers	Generic	Working
	DaVinci	Working
	User defined templates	In Progress (new)
Data selection	Connection to bookkeeping database	Working
	Creation of XML slice	Working (new)
Installation	AFS	Working
	Local	Working
Job management	Splitting of input data	In Progress
	Merging of output	In Progress (new)
Roaming	Job profile	Deferred
	Options and DLL packaging	Working (new)
CLIP	Integration with GANGA core	Working (new)

Conclusion

New Ganga-3 release made

Supports features required for proof of concept for LHCb distributed analysis.

First implementation of scripting through CLIP.

Many new LHC*b* use cases supported.

Last release in old framework.

Well on the way for refactored release

Very dynamic way of working at the moment.

Expect to have a version with same functionality as Ganga-3 by mid April.