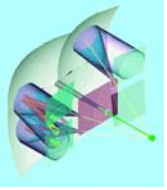

Imperial College London

Summary information:
The Ganga/User perspective

Robert W. Lambert, Imperial College London.

On behalf of the GangaLHCb team, and the typical Ganga user

User Motivation

1. Managing data split into many jobs
2. Analysing problems on the GRID
3. Calculating luminosity (similarly for real data)

- All require parsing of stdout
- This is problematic because:
  - Many ‘typical’ problems can be misinterpreted
  - The stdout can be very confusing
  - Certain summaries require an expert
  - Very time consuming
- Users have written ad-hoc complex scripts to do this

Root Cause

- Stdout:
  - not a robust method of documentation
  - not a ‘safe’ language
  - not easily parsed
- The Gaudi job knows the information already
- Summary information from the Gaudi job would be ideal
- Usage in Ganga:
  - From Local/Batch/Grid etc. etc.
  - Ganga can support intelligent handling of the summary info
  - Summary info can be added/catted together
  - Simplified job debugging/resubmission

What information

1. “Application manager Finalized successfully”
2. File and event information:
   - [filename, #events, statuscode] (read, partial, fail, not read)
3. Luminosity information:
   - Whatever numbers are needed to normalise/calculate the lumi
4. General counters
   - Summary of Gaudi `counter("string")` methods
5. Total memory useage
   - Maximum image size

Constraints

1. Simple, Computer and Human-readable
2. Well-defined format
3. Stable implementation format
4. For the user:
   - Much smaller than the stdout
   - Easily parsed
   - Should be optional

➢ Very loose constraints!

➢ Preferred: XML **with a well-defined schema**

Idealised use case

- Get the summary information from any gaudi job

```
DaVinci().XMLSummary='summary.xml'
```

- User-readable, manipulations within Ganga

```
j.merger=XMLMerger(files=['summary.xml'])
s=j.application.summary()

ratio = s.luminosity() / j.inputdata.luminosity()

k=j.copy()
k.inputdata=s.data(status=['fail','unread'])
k.submit()

eff = s.counter['MySuccess']/ s.counter['MyTotal']
```

Backup

- Backup slides hereafter

Case 1

- Running on MC data in many jobs
- The user wishes to know:
  - How many events were run over?
  - Which files succeeded/failed/skipped?
- Right now the user must:
  - Parse the stdout, or personally add histograms
- This is problematic because:
  - The stdout is not ‘standardised’
  - The user has to check that everything finalised correctly
- Users have written ad-hoc complex scripts to do this

Case 2

- There has been a problem running on the grid
- The user wishes to know:
  - Was it due to their own algorithm?
  - Is the data still fine to merge?
  - Which event caused the problem?
- Right now the user must:
  - Parse the stdout, eventually rerun in debugging mode
- This is problematic because:
  - Many ‘typical’ problems can be misinterpreted
  - The stdout can be very confusing
- Typically the user emails the mailing list for help

Case 3

- Running on real data
- The user wishes to know:
  - What is the integrated luminosity?
  - Was that the maximum luminosity I could have obtained?
- Right now the user must:
  - Interpret the Luminosity info in/for each DST (Bookkeeping)
  - Parse the stdout, to find the files+event counts
- This is problematic because:
  - It is an expert procedure
  - It is very time-consuming
- The scripts to do this have not yet been written