



TM and © Laurent de Brunhoff

# Kanga Data Distribution and Management

Tim Adye

Rutherford Appleton Laboratory

Computing Model 2 Implementation Workshop

Rutherford Appleton Laboratory

13<sup>th</sup> January 2003

# Talk Plan

- Existing Kanga distribution
  - Kanga file distribution [ +Alessandra Forti]
  - ftp tools
- Existing Kanga file management
  - Using multiple file-systems [ +Terry Hung]
  - Checking
  - Archiving [ +Alvise Dorigo]
- Ongoing developments
  - Automatic check and delete at SLAC
  - Deep copy [ +Fabrizio Salvatore+Ulrik Egede]
- Summary: key features of this system

# Kanga distribution control

- File import at each site is controlled by the local skimData database
  - This is mirrored nightly from SLAC or other Tier A
  - complete copy of the skimData tables, maintained by **skimSqlMirror** (see Alessandra's talk)
    - Includes all datasets not just those needed locally
- Identical information on each file, except
  - **import\_status** field: local file status
    - one 2-character entry per file
  - **skim\_archive** table: file info in mass-storage system
    - one row per archived file
- Kanga Conditions files are handled separately
  - Simple **rsync** of the directory

# import\_status

- **import\_status** field takes the following values
  - 2** new entry, just created by **skimSqlMirror**
  - 1** file is not required at this site
  - 1** file is to be imported
  - 1A, 1B,...** file is to be imported with priority **A, B,...**
  - 0** file is on disk
  - 0C** file is on disk and locally checksummed
  - 0B** file is on disk and backed up to tape
  - B** file is just on tape
  - E** file has been deleted
- By default, skimData only sees files with **import\_status=0\***
  - override with **--remote=X**

# File Import Steps

1. New Kanga file production completed at Tier A and marked "OK" in database
2. Cron job at Tier C performs the following actions...
3. **skimSqlMirror** copies new database entries
  - Gives **import\_status=2**
4. **skimSqlSelect** selects files for import
  - Starts with all files with **import\_status=2**
  - Selections use skimData commands to define required streams etc
  - **import\_status=-1, 1A, 1B, etc**
5. **skimImport** transfers files from Tier A
  - a) Uses all files with **import\_status=1\***, sorted by priority+job+stream
  - b) Directories and symbolic links (symlinks) created
  - c) File transfer program (**scp, bbftp, bbcp,...**) executed
  - d) If ftp successful and file size is correct, then set **import\_status=0**
6. File now visible by skimData and available for use

# ImportUtils

- Perl module used by **skimImport** provides a **common interface** to different file transfer tools
  - Currently supports **scp**, **bbftp**, and **bbcp**
    - Could add **GridFTP**, **rsync**, **Unix ftp**, etc
    - **OO** interface and **modular** design allows easy addition of other tools
  - Provides some “missing” functionality for different tools
    - Creates temporary control files where necessary
    - Where possible, uses a single command for multiple files
      - start and stop time quite significant for smaller Kanga files
      - Does not yet work for **bbcp** (requires **bbcp** bug fix)
    - Automatic directory creation (GET only)
    - Hide and protect files during transfer (GET only)
  - Performs file allocation and symlinking onto multiple filesystems

# Multiple File-systems (1)

- All Kanga data access is (currently) via NFS
- Collection name directly maps to a single path name

- Eg.

`/groups/SP/0050/4300/M8.8.1aV01x15F-R8.8.1aV01/  
00504384/Kanga1010cP1/P0000/KanGA/SPKanga`

becomes

`$BFROOT/kanga/EventStore/  
groups/SP/0050/4300/M8.8.1aV01x15F-R8.8.1aV01/  
00504384/Kanga1010cP1/P0000/KanGA/SPKanga-micro.root`

- This path name refers to a single directory tree
  - At larger sites, this directory tree contains symlinks to directories/files on different file-systems

# Multiple File-systems (2)

- Allocation of data between file-systems is part of Kanga production and import procedure
  - Kanga production allocation is SLAC-specific (**dynamicBalancer**)
  - Import allocation defined by a configuration file
    - Different sites use different organisation, eg.  
SLAC:  
`groups/SP/0143/2000/M10.3.1aV01x20F-R10.3.1aV01/ →`  
`/nfs/farm/babar/kanga28/groups/SP/0143/2000/M10.3.1aV01x20F-R10.3.1aV01/`  
RAL:  
`groups/SP/.../SPKanga-micro.root →`  
`/stage/bdata-data29/kanga/EventStore/groups/SP/.../SPKanga-micro.root`
  - Both allocate on file-system with most free space
- Automatic reorganisation of existing file-systems is currently SLAC-specific (**staticBalancer**)

# Multiple File-systems - to do

- x Need site-independent **dynamicBalancer**, **staticBalancer**, and similar tools
- Different file-system allocation algorithms?

# Checking

- **kangaCheckFiles** performs the following checks
  - Looks for db files not on disk
  - disk files not in db
  - compares file sizes and checksums
  - Shows summary information
    - eg. total disk used by different datasets
- Checksum is filled in db at SLAC, mirrored elsewhere
  - Currently just RAL and Karlsruhe
  - Separate step, perhaps in parallel with export and other use
  - **cksum** command can be automatically run on file server
    - Requires ssh access to NFS server

# Checking – to do

- x Only a few db-internal consistency checks
- x Could have a more thorough symlink and file-system checks
  - eg. files with no symlink from index tree
- x No automatic fixing for common problems

# Archiving

- **skimSqlselect** can be used to select files to be archived
  - Similar to import selection, but using `import_status=0S`
- **skimBackup** can archive individual files or pack together into **tar** files
  - Most mass-storage systems prefer files 0.5-2 TB
  - Different datasets (release, stream, etc) go into separate **tar** files, ordered by production job number
    - Hopefully clustering useful when staged back
- Parallel tape transfers handled automatically
- Site-specific backup and check procedures
  - SLAC, RAL, and CASPUR implemented

# Archiving – to do

- x Simplify running multiple parallel backups on different CPUs
  - eg. using batch system
  - or even run **tar**/tape write direct on file server?
- x This is a write-only archive!
  - Single files can be read back by hand
  - Need tools to read back many files when required
    - eg. disk crash
  - Will need automatic on-demand staging system
- Do we need to pack files together (**tar**) in future?

# Developments: SLAC exports

- Currently file deletion is initiated by hand
  - Standard tools do help: **kangaCheckFiles** and **skimDelete**
- If production site (eg. SLAC) is not to be the primary analysis site, may want to delete files that
  - have been archived to tape
  - have had checksum written to database
  - have been imported to other Tier A (eg. RAL)
  - have had the RAL copy checksummed
  - have been on disk long enough for nearby Tier C sites to import direct if they wish (otherwise they can use RAL)
- This can all be done with existing tools
  - Need a little glue (and the courage!) to initiate automatic deletion

# Developments: deep copy (1)

- (Just) starting to produce pointer skim files at RAL
  - 104 files for each run point to one AllEvents file
- Tier C sites cannot use these without copying AllEvents file too
- Deep copy procedure converts pointer skim file to data skim file
  - Data skim file can be used at Tier C without AllEvents file
  - Output can be written direct to Tier C disk using ROOT Daemon (**rootd**)
  - Will soon integrate into **skimImport** as another “ftp” protocol

## Developments: deep copy (2)

- “On-the-fly” deep copy initiated from importing site
  - Just like current **skimImport**
- File selection and local file-system management can use existing **skimImport** architecture
- **ImportUtils\_root.pm** (to be written):-
  1. start private local **rootd** using random password
    - There must be a better way, but this should be reasonably secure (password only passed encrypted)
  2. ssh to Tier A and start **RooskimUtil**
  3. Send **rootd** password and specify input and output files
    - Input: pointer skim files on Tier A disk
    - Output: data skim files, written to Tier C disk via **rootd**

# Summary: key features (1)

1. Each site has its own copy of the skimData catalogue.
  - Apart from daily updates from Tier A, it doesn't know what is at any other site.
    - Once copied, no access to Tier A required
      - simple for laptops!
    - Complicates Grid data location
      - but Alessandra has a solution 😊
  - Imports are entirely driven by the import site
    - No database access required to the export site
    - Fewer firewall, security, and contention problems
2. Only small differences between databases at different sites
  - Just in **import\_status** field and **skim\_archive** table

## Summary: key features (2)

3. Collection names and TCL files are site-independent
  - modulo a few global parameters
4. Trivial mapping between collection namespace and directory tree
  - Mapping to different file-systems done using symlinks.
5. Import selection is done at the file level and is decoupled from the file transfer procedure
  - Single file selection not much used by users, but very useful for error recovery.
6. No separate pre/post-processing of import files
  - even with skim deep copy