Farming Software

Ping Yeh

CDF Internal Farms Review
September 2, 1999

Outline

- Strategy of Software Development
- Requirements and Analysis
- General Design
- Choice of Implementation Language
- Specification (ongoing)

Strategy of Software Development

- Explore Requirements
- Conceptual Model, System Analysis
- General Design, Test of Concepts
- Specification, Test of Technology
- Implementation, Test of Codes

Reference: CDF note 4997

Requirements on the Software

- ► Excellent bookkeeping and catalog
- ► Throughput: 20 MB/s in, 11 MB/s out
  Capable of handling peak level 3 rate at 75Hz
- ► Flexibility
- ► Scalability
- ► Minimal impact of worker node failures
- + No run section gap
- ► Compatibility with other softwares {CD,DH,L3}
- ► Code Maintenance

Conceptual Model

In building the conceptual model of the production farm, we focused on

- Data movement
- Identifying tasks
- Identifying resources
- Synchronization issues

Conceptual Model Diagrams

Show 2 diagrams here

- Conceptual Model (the diagram with faucet and sink)
- Activity Diagram

System Analysis

- Pipeline and buffers
- Parallel processing and synchronization
- Throughput analysis
- Clean worker node (minimal impact)
- Bookkeeping and catalog
- Error handling
- Farmlets (flexibility)

Pipeline and Buffers

- In the most simplistic view, the farm system is a pipeline with 7 stages:
  stage, dispatch, reconstruct, collect, split, export, spool
- Three disk buffers are called for:
  - `>=500 GB` raw data
  - `>=200 GB` unconcatenated inclusive pads
  - `>=800 GB` concatenated pads
    - Will likely be more
- Tape library with robots is our largest buffer

Parallel Processing and Synchronization

- Data - Database synchronization before staging
- Run section gap in concatenation
- Export and spool in parallel?
  - Farm pushes "express" dataset to external disks. All datasets are checked by exporter, export or not, before being moved to spool area.
    Avoids potential synchronization problem.
  - Alternative: no export. DIM handles spool area. Analysis machine pulls data via DIM-DIM interface
  - Need more input from physics group to decide

Throughput Analysis

- Throughput = hardware speed x efficiency
- Throughput of the whole pipeline is determined by the slowest stage
- Factors:
  - Idle time
  - Error rate
  - Error recovery time
  - Number of retries

Efficiency

- To achieve 70% efficiency:
  - Idle time < 20% of processing time
  - Error rate < 10%
  - Error recovery time < processing time
  - Retry at most 3 times

Clean Worker Node

- To fulfill the requirement of minimal impact from worker node failures
- Output files of the reconstruction job must be moved to the output node soon

Bookkeeping and Catalog

- Catalog is being done by Data Handling group (Eric Wicklund)
- Bookkeeping:
  - What, where, when
    to allow reproducibility and performance analysis
  - Use available database software
  - Three databases: Raw data, Physics data, Failures
  - Current plus Historical

Error Handling

- Error prevention during design and implementation (hardware and software)
- Error detection as much as possible
- Four levels of error handling
  - *In situ*
  - Coordinator (see software structure)
  - Shifter
  - Expert
- Requirments on the CDF executable follows

Requirements on the Executable

- Bookkeeping: production executable version in addition to module versions.
- Error handling:
  - Ability to gracefully terminate when error occurs, gives error report and last successful event in log file.
  - Start processing from a specified event in a file
  - Append output file
- Static link

Flexibility Requirement

- Add/remove/change streams or datasets, change priorities, without shutting down the farm
  - Reprocess data
  - Generate Monte Carlo sample
- Concept of Farmlets:
  A logical division of the farm into sub-farms

Farmlets

Defined in terms of

- Streams/filesets/files to process
- Executable and template input file for reconstruction and splitting
- Archiving policy
- Allocated resources
  Tape Drive, CPU, bandwidth, Disk

Resource Sharing Mechanism

- Logical division
- Sharing policy: resource by resource

```
[farm]
farmlet=fresh,reproc

[input drives]
# fresh owns drive1 and 2
drive1-2=fresh

# shared when conflict, 30% vs 70%
drive3-5=fresh:3,reproc:7
```

General Design

- Design principles
- Software structure
- General design of each subsystem

Design Principles

- Error prevention and error handling
- Scalability
- Maintenance

Software Structure

- System Decomposition
- Class Hierarchy
- Crash Recovery

System Decomposition

- Three-tier Design
  - Interface tier
    Talk to external systems and user
  - Business tier
    Carry out the logic to get things done
  - Data tier
    Deal with data and resources

Interface Tier

- Human Interface:
  Farm Monitor and Control,
  Messenger
- Data-File Catalog Interface {DH}
- Database Interface {DB}
- Operator/Robot Interface {CD}

Business Tier

- Coordinator
- Database Checker/Retriever {DB}
- Stager {DH:DIM}
- Dispatcher
- Reconstructor
- Collector
- Splitter
- Exporter
- Spooler {DH:DIM}

Data Tier

- Resource Managers
  - Disk Manager {DIM, simple one for inc. pad. disk}
  - Tape Manager {DIM? FBS?}
  - CPU Manager {FBS}
  - Bandwidth Manager {FIPC/FBS/...}
- Bookkeeper
- Internal Messenger

Class Hierarchy

Preliminary

- `Subsystem`: basic building block
  do, cleanup, ...
  Daemon, Stager, Spooler, DBChecker
- `Daemon`: inherited from Subsystem
  start, pause, resume, shutdown, kill, recover
  Coordinator, Dispatcher, Splitter, ResMan
- `ResMan`: inherited from Daemon
  submit, status, cancel, suspend, enable, disable
  DiskMan, TapeMan, CPUMan, BWMan

Daemon Subsystem

- Perform input/output in the background while listening to commands
- Responding to commands is of higher priority than working on the input/output
- Behavior:
  - Spend most time listening to command channel
  - Look at input quasi-periodically
  - Spawn a child process or start a thread to handle the input/output to minimize response time

ResMan Virtual Class

Preliminary

- Manages one resource pool, e.g., tape drives
- Create one queue for each farmlet at start-up
- To allocate a resource, a farmlet user submits a job script with resource name $RESOURCE
- When a resource is available, ResMan checks entries of all queues and grant the resource to an entry according to resource sharing policy
- The job script is started by ResMan with $RESOURCE environment variable appropriately defined

Command Channel

Implementation options of the command channel:

- ► Front-end
  A front-end program receives command through its command line arguments, and relay the command to the daemon process through a private IPC
- ► Direct inter-process communication (e.g. socket)
- ► FIPC developed by Computing Division

	Front-End	Direct	FIPC
couplings	less	more	middle
speed	slower	faster	slowest
availability	prototype	prototype	Fermi product

Crash Recovery

- Initiated by coordinator, which sends "recover" command to doers and resource managers
- Doers rely heavily on bookkeeping to resume to the state prior to crash
- Queues of resource managers are persistent, jobs that were running are re-started by resource managers
- Uncomplete files left around on worker nodes are cleaned up

Implementation Language

- Python
- Features:
  - Object-oriented scripting language
  - No compile/link, fast development and prototyping
  - Abundant and easy-to-use interfaces to
    - operating system: socket, regexp, ... etc
    - other packages: Tk, most database servers, ... etc
  - May replace components by C codes for efficiency
  - Nice error handling scheme:
    `try - except - else`
  - And much more...