Service data from the SAFE

Stephen Booth

Introduction

- SAFE in the user administration system developed by EPCC
  - Java JSP/Servlet
  - Data held in relational database.
- It includes:
  - User registration
  - Helpdesk
  - Resource management
- Report generation and accounting is an important component of this.

Requirements

- Performance
- Flexibility
- Extensibility

Performance

- Need to handle large numbers of records.
  - Reports that span several years worth of data quite common.
  - A Large HPC system can run several million jobs In that time.
  - Hector database > 2.6 million records
- Even harder for systems with a throughput workload.
  - Typically many smaller jobs
  - Total record count can be several orders of magnitude higher.
  - ECDF (Edinburgh University local cluster) > 50 million records

Flexibility

- Want general purpose reporting system
  - Re-use same system for :
    - Job usage reporting
    - Disk reporting
    - Allocations
    - Helpdesk
- Need to support many different types of accounting data.
  - Different batch systems
  - Unix process level accounting.
  - Grid-middleware
  - Job starters (e.g. ALPS)
  - Etc.

Common data formats

- We chose *NOT* to map all data to a common internal data format. We do *NOT* require data to be uploaded in a standard format.
- Upload data in whatever format is convenient
  - E.g. PBS accounting logs
  - Keep upload scripts as simple as possible, for text based formats we use the same script everywhere.
  - For systems like SLURM no text accounting log so need slightly more work but still easier than mapping to standard format.
  - Moves code complexity to central system where it is easier to re-use code
- Store each data source in its own database table.
  - Table schema usually reflects format of uploaded data.

Common data view

- Still need a common view of data to generate unified reports.
- We generate this dynamically not at data upload.
  - Less risk of data-corruption/information-loss if data stored in close to original format.
  - Easier to extend the system while retaining backward compatibility with old databases.

Extensibility

- General reports address well understood metrics and change little over time.
- Also useful to be able to add new reports and explore new metrics.
  - Add new reports dynamically to live system.
  - Custom reports for particular groups of users.

Data model

- The reporting sub-system uses a property based abstraction.
  - Each record is viewed as a collection of key-value pairs.
    - Similar to approach used in many scripting languages
  - Highly flexible abstraction as the set of properties is not explicit so same code can be used for many different types of data.
- Need to worry about performance
  - Simplest approach would read individual records into memory for processing.
  - Want to use high level SQL operations where appropriate.
  - Need to be able to map property abstraction onto SQL.

Factory classes

- Records are viewed as collections of properties managed by a factory class.
  - Factory can supply registry of supplied properties, these also specifies additional meta-data such as type bounds.
  - Additional properties can be defined as expressions over other properties
- Factory classes roughly correspond to database tables
  - Implement methods to perform high-level queries that can be mapped to SQL
- Reports can access any table/class that implements the required interfaces
  - Helpdesk/Accounting reports both use the same sub-system
  - Composite factories allow queries to multiple tables.

Property Expressions

- New properties can be defined as expressions over existing properties.
- We use this to define the common data view for report generation.
  - a common set of properties independent of underlying data source.
- Examples:
  - WallClock = EndTime – StartTime
  - Residency = Numnodes * WallClock
  - SlowDown = (EndTime–StartTime)/ (EndTime-SubmitTime)

Reference properties

- Properties are strongly typed:
  - Number, String, DateTime, Reference
- Reference properties are pointers to records in other tables.
- Property expressions can access properties on the remote record.
  - Charge = Residency * Machine[ChargeRate]
  - When mapped to SQL this implies a JOIN to the remote table.

SQL mapping

- Property expressions need to be mapped to SQL fragments.
- Internally expressions stored as abstract-syntax-tree objects.
  - Visitor pattern used to implement operations.
    - Evaluate expressions
    - Create SQLValue
    - Create SQLExpression
- SQL fragments represented as SQLValue objects.

  Implement methods to:
  - Add fragment to SQL statement
  - Create value from java ResultSet
  - Optionally provide SQLFilter to modify FROM and WHERE SQL clauses.

SQLExpressions

- Sub-types of SQLValue
  - SQL fragment is a single expression equivalent to value produced.
  - Can be combined at SQL level.
- Required for SQL reduction operations
  - MIN, MAX, SUM etc.
- In other contexts SQLValue is sufficient

Non SQL operations

- Also possible to implement properties using Java fragments with no SQL equivalent.
- Reporting code falls-back to iterating over objects.
- Only needed in special circumstances.
  - Elapsed working hours for helpdesk reporting.

Parsers/Policies

- Data is imported using a combination of plug-in parsers and policies.
- Parsers
  - Split input into separate records
  - Parse record to set of properties
  - Specify properties to uniquely identify record.
    - Used to detect pre-existing records in database
- Policies
  - Generate additional properties
    - E.g. Charge, Reference properties
  - Apply side effects
    - E.g. Decrement budget allocations.

Aggregated records

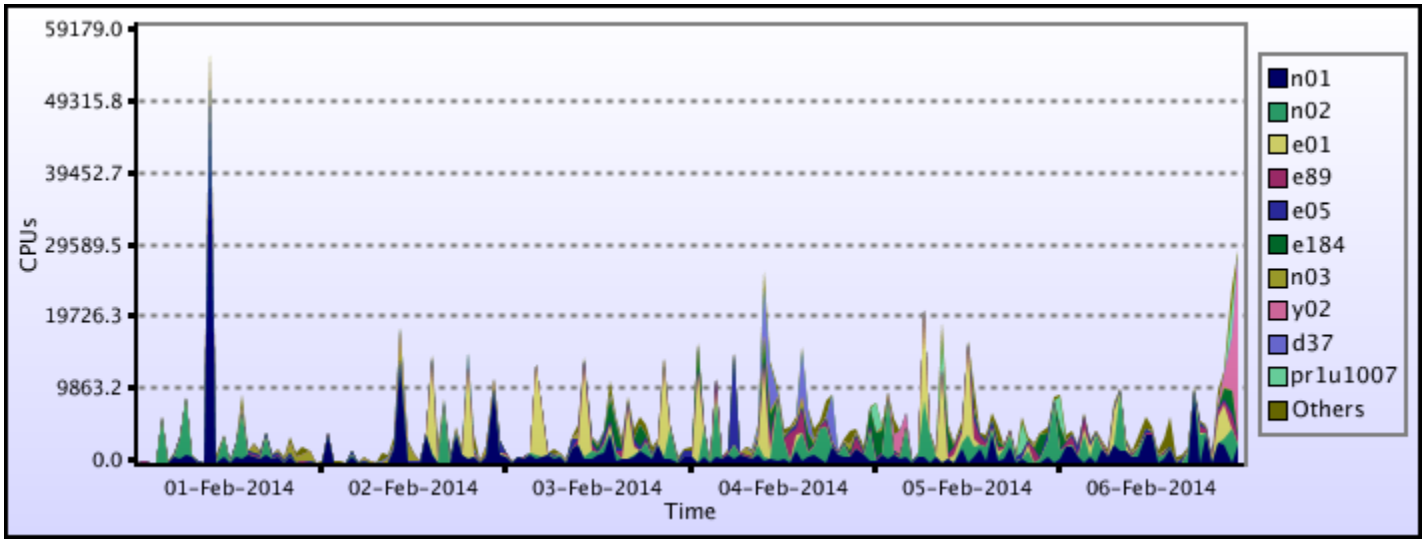
- Policies can be used to build tables of aggregated records.
  - Records with similar properties are merged into aggregates
  - Time properties are rounded to hour or day boundaries.
- Un-aggregated table still exists if required.
- Aggregation reduces number of records that need to be processed.
  - Particularly effective on throughput systems where many records are very similar.
- Aggregates are created/modified as new data is parsed.
  - Can also re-generate aggregates from original table.

Time mapping

- SAFE provides special support for records that overlap ends of the time periods.
  - For long time periods it is sufficient to select records based on a single time property (e.g. completion time of job).
  - For short time periods (comparable or shorter than the job length) this gives a distorted view of the data.
  - This is a particular problem for graphs showing evolution against time as each point on the graph corresponds to a short time period.

Naïve Time graphs

- If we plotting records that finished in a time period.
  - Graph depends strongly on choice of plot periods.
  - Graph is also difficult to interpret
    - Example shows average size of job completing in each period.



Weighting functions

- Records that overlap ends of time period need to be scaled.
- Two distinct types of scaling are needed depending on nature of property being.
  - Accumulating properties
    - Values that accumulate during record
      - CPU-Time,
      - Wall-clock
      - Charge
  - Instantaneous properties
    - Values that can be measured at a particular instant in time
      - CPUs
      - Memory used

Accumulating properties

- Need to be weighted by fraction of record that overlaps with the period

$$V_{period} = V_{record} \frac{l_{overlap}}{l_{record}}$$

- Values may be summed to give value accumulated in period.
- Divide by period length to give an average rate.

Instantaneous properties

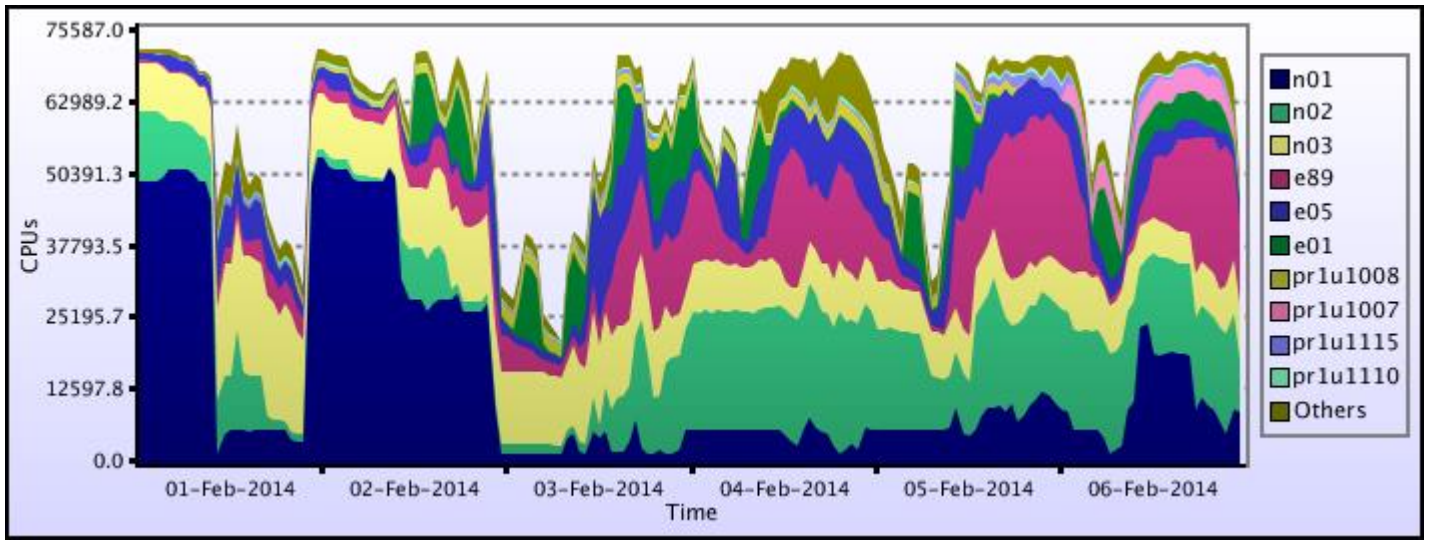
- Value at any given instant is the sum of values from records that cross that point in time.
- Representative value for the period is the time average
- Weight by fraction of *period* that is overlapped and sum over records.

$$V_{period} = V_{record} \frac{l_{overlap}}{l_{period}}$$

- Can turn instantaneous property into accumulating by multiplying by record length
  - Time average is the same as the rate calculated from this property.

Time graphs

- Time averaged CPU plot
  - At this resolution many jobs cross multiple time periods.
  - Different division into plot periods would give broadly similar plot.



Time bounds

- Need to choose two time valued properties to denote start and end of record.
- This may be different depending on quantity.
- CPUs used would use *job-start* and job-end
- Jobs waiting would use *job-submit* and *job-start*

Template reports

- Reports are generated using XML reporting language.
  - New reports can be added dynamically to running system.
- Template expanded in multiple stages using XLST transforms (plus extensions to perform queries)
- Final stage maps XML to desired output format
  - HTML
  - PDF
  - XML
  - CSV

Features

- Access control
  - control access to reports or report sections.
- Parameters
  - form generation and parameter expansion.
- Filters
  - select records equivalent to SQL WHERE clause.
- Single-value-queries
  - results placed in-line in text.
- Tables and charts
  - Use high level operations implemented by Factory classes.
- Formatting
  - Format individual records, useful for generating job listings or generating interchange records

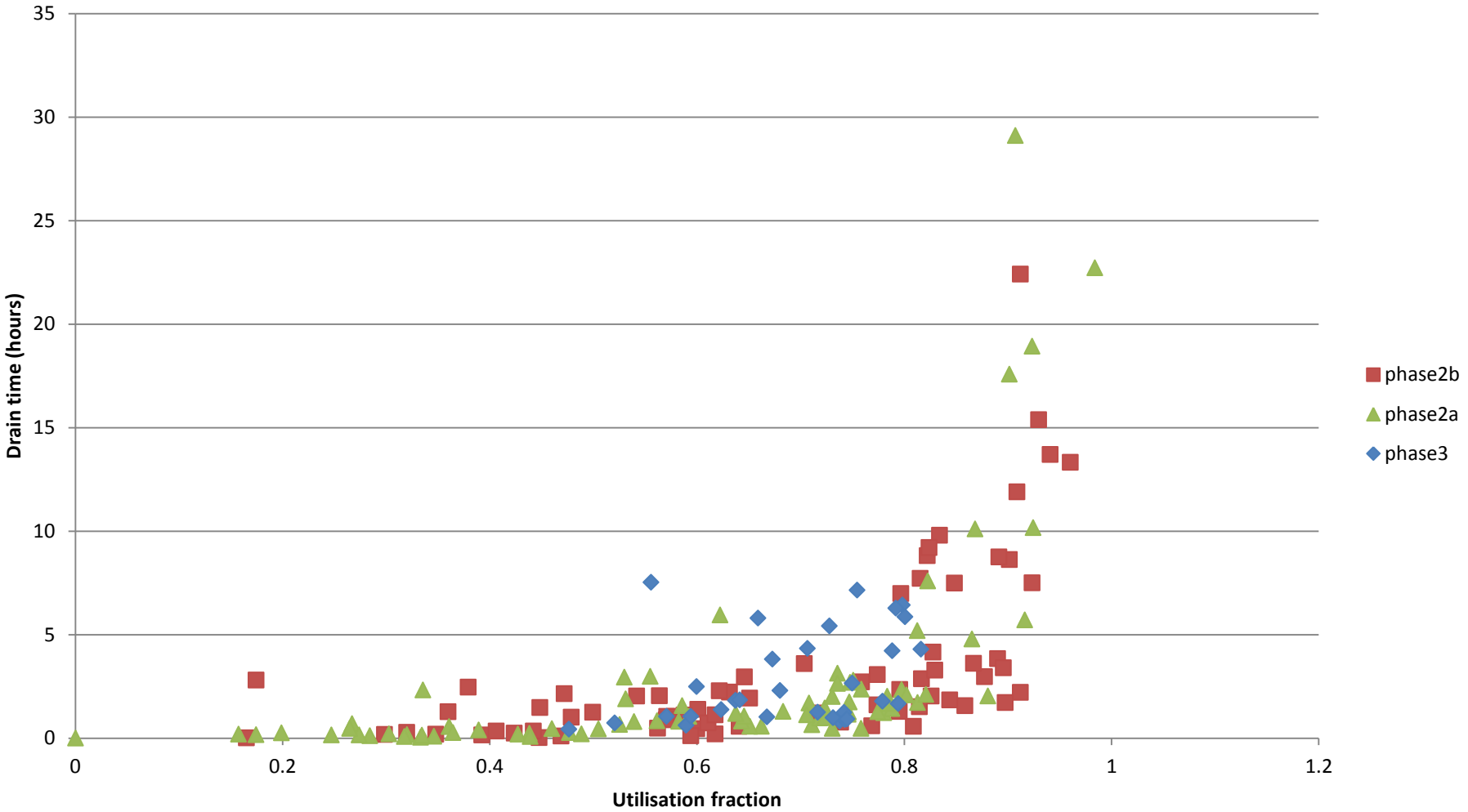
Custom reporting

- XML reporting useful for most general tasks.
- Can also access reporting code programmatically for more complex analysis.

Utilisation vs drain time

- Looking for correlations between machine utilisation and job waiting.
- In practice wait times depend on size of job. Ideally want metric that looks at total amount of queued work.
- Used weekly average to remove cyclic variation.
- Plotted time average of machine utilisation against time average of work waiting.
- Remarkably consistent picture across 3 hardware phases of the HECToR service.

Utilisation vs drain-time



Project use profile

- Looking for profile of use across lifetime of projects.
  - Projects all start and end at different times and have varying lengths.
- SAFE has unified database so project start/end and allocations are available as well as usage data.
- Calculated percentage of allocation used at various fractions through project lifetime.
  - Plotted percentiles to give overview of use-profiles.
  - Note not all projects had finished so less data towards project end.
- Some fraction of projects don't even start work until half way through the project.

Project use profile