ECL – Implementation

Richard Chapman

VP R&D LexisNexis Risk Solutions

Overview

- ECL Compiler
  - generates shared objects representing ECL code
- ECL Execution Engines
  - Single node – Eclagent
  - Multi-node ETL – Thor
  - Multi-node online - Roxie
- Other system components
  - Middleware, system data store, scheduler etc

ECL Compiler

- Parse ECL
  - LALR(1) using bison and flex, to create expression tree
- Optimizations
- Transformation
- Generate C++ and XGMML
- Compile and link C++

Parsing

- Fairly standard LALR parser
- Lexer state modified according to what symbols are acceptable, to reduce namespace pollution issues
- Case sensitive language would have helped!
- Syntax check stops at this point

Optimizations

- Standard optimizations
  - CSE
  - constant folding
    - including library calls
    - including dataset expressions
    - including tracking constants into expressions
      - e.g. IF(a, 'b', 'c') IN ['a','c'] => NOT a
    - Including tracking constant field values in datasets

Optimizations contd

- ECL optimizations
  - CHOOSEN(SORT()) -> TOPN
  - SORT(SORT()) -> SORT
  - Field trimming
  - FILTER(SORT()) -> SORT(FILTER())
  - Tracking sort order and distribution
    - Can avoid resorting/redistributing
    - Can make global operations local

Transformation

- Declarative “what” into imperative “how”
- Create execution graph
  - Add splitters
  - Check dependencies
  - Check resourcing
- Extract invariant expressions
- Select activities
  - Dependent on target engine capabilities
  - Use compound activities

Generation

- C++ code generated via peephole optimizer
- XGMML representing activity graph generated as resource file
- System c++ compile/link invoked to create shared object

Compiler notes

- Actually a bit more iterative than the simplified description above…
- Declarative language much easier to transform
- Remote compiler support using “archive” generation

Execution

- Standalone executable
- EclAgent – single node
  - Very much like standalone…
- THOR
  - ETL engine using 100s of nodes in parallel
- Roxie
  - Online query engine using 100s of nodes

EclAgent

- Load XGMML
- Instantiate activity per node
- Connect them up according to the edges
- Use dynamically resolved helper functions to control activity behavior
- EclAgent has limited multi-threading support
- Launched on demand, as many processes as you want.

Thor

- Master node co-ordinates and marshals
- Slave nodes do (almost all) the work
- Master and slaves all load XGMML, create activities etc
- Local activities execute on slaves without master involvement
- Global activities use master to co-ordinate
- Master controls graph start/stop

Thor (contd)

- Global SORT
  - All slaves sort locally, propose split points
  - Master proposes combined split points
  - Slaves indicate resulting distribution
  - Iterate until skew is acceptable
  - Distribute using agreed splitpoints
  - Merge incoming data streams
  - See patent for details ☺

Thor (contd)

- Global JOIN
  - Sort LHS, co-sort RHS, then walk group by group
- HASH JOIN
  - As global join, but hash rather than sort to get rows together
- LOOKUP JOIN
  - All slaves pass RHS hash table to master
  - Master distributes combined hash table
  - Each row in LHS looked up in resulting hash table
- Also ALL JOIN, SELF JOIN, sliding window JOIN…

Thor (contd)

- Reasonably multi-threaded (and getting more so)
- Multi-thor can be used to increase CPU usage further
- ‘Folded’ multi-thor also in experimental use
- Traditionally, uses local storage
- Experimenting with off-node storage

Roxie

- Multiple servers and multiple slaves
- All queries preloaded for repeated execution
- Servers use multicast to send queries to relevant slave channels
- Slaves use local disk to fulfil requests, and return results
- Extremely multithreaded

Roxie (contd)

- Multi-level b-Tree indexes
  - Top level index tells us which channel to use
- In-memory indexes
  - Can use ‘broadcast’ to get answers from all channels
  - Also useful for ‘noroot’ indexes

Roxie (contd)

- Smart stepping
  - Propagates through filters, etc
  - Effective in a dynamically created graph
  - N-way join shares information between the index reads that feed into it

Other components

- ECL Watch and Graph Viewer
  - An experienced ECL programmer will think in terms of the execution graph as much as the code they write
- Output visualizations
  - Traditional row/column not always the best!
  - Also can display as a graph of connected vertices.
- Query Debugger
  - Breakpoints on edges in the execution graph
  - Display row data, break if condition met etc. etc.

Future

- More dynamic resourcing
- Merge Roxie and EclAgent
- ECL generators
- Graph manipulation enhancements
  - Allow rows to exist more than once?

LexisNexis