White Paper

HPCC Systems: Enterprise Control Language (ECL): An Overview

Author: David Bayliss, Chief Data Scientist

Date: May 21, 2011

LexisNexis®

Table of Contents

Executive Summary 3
What is the Enterprise Control Language? 3
Tackling the Declarative Language Problem 3
How Enterprise Control Language Works 4
Learning ECL 5
Comparing ECL to Other Solutions 5

Executive Summary

Enterprise Control Language (ECL) is the query and control language developed to manage the HPCC (High Performance Cluster Computing) and truly differentiates it from other technologies in its ability to easily and efficiently provide flexible data analysis on a massive scale.

This white paper explores ECL, how it works, how it tackles data problems, and how it's an easier programming language to use compared to other solutions.

What is the Enterprise Control Language?

The lynchpin of the HPCC is the Enterprise Control Language as it is the key that allows programmer and hardware alike to function optimally. ECL is best described as a heavily optimized, data-centric declarative language. Exactly what that means is detailed below; but the essence is that it is a **language specifically designed to allow data operations to be specified in a manner which is easy to optimize and parallelize**. It is also designed to allow multiple programmers to work on an ever growing array of data and maximize the leverage obtained.

A reasonable, if informal, definition of a declarative language is that it is one in which you specify what you want done rather than how to do it. The most distinctive feature of them is that they are extremely succinct; it is common for a declarative language to require an order of magnitude (10x) less code than a procedural counterpart to specify the same problem. This seems like a nirvana and for a while, declarative languages were touted as 'fifth generation' and the 'languages that would put programmers out of business.'

But a problem arose; while a declarative language relieves the programmer of figuring out 'how to do it," someone still has to do the figuring. Generally that was left to the compiler writer, and as compiler writers were in short supply, the languages could only be extended to those problems that the compiler writer had already figured out. Many refer to this as 'The Declarative Language Problem'. Thus declarative languages tended to be restricted to very narrow fields of highly specialist work and rapidly fell from fashion.

Tackling the Declarative Language Problem

ECL tackles the Declarative Language Problem three ways.

Firstly it is **data centric**. It has restricted itself to those problems that can be specified by some form of analysis upon data. It has defined simple but powerful data algebra to allow highly complex data manipulations to be constructed.

Secondly it is **extensible**. In fact 'writing in ECL' and 'extending ECL' is the same thing. Precisely what one does is define new terms (called attributes) which then become a part of the language. Therefore a new ECL installation may be relatively narrow and generic in scope but as it is used its abilities expand to allow new problems and classes of problems to be stated declaratively.

Thirdly it is **internally abstract**. The ECL compiler generates C++ and calls into many 'libraries' of code, most of which are major undertakings in their own right. By doing this, the ECL 'compiler' is machine neutral and greatly simplified – allowing the compiler writing resources to focus on the most important thing – making the language good.

For those familiar with computer languages, our third item above may have been startling; how can we be internally abstract and yet still claim to be highly optimized? Is not optimization all about going down to the metal? It turns out that in the world of Big Data the answer is no. At least the problem of going down to the metal can be left in the capable hands of a good C++ compiler. The real performance gains come from the ability to distribute data and work optimally across a collection of processing nodes, which is easy to state but very hard to do.

In practice, there are three major areas one needs to tackle:

1) **Minimization of data and data movement**. When working with Big Data one may assume the data is big, thus moving it is nearly always a mistake. In fact, even keeping the data in an active process might be a mistake if it can be avoided. Thus the ECL compiler spends a lot of time looking at all of the data required by a process and plotting which data can be dropped when and how to get to the end result with a minimum number of data movements. By far the biggest win compared to an SQL system is that the code moves to the data; not the data to the code. When compared to a Hadoop system the win is that the forced distribute in the MapReduce pair does not usually have to happen.
2) **Doing work once**. Given that the very process of touching (reading from disk, keeping in memory, etc.) an item of data is expensive, it is important to common up all the different places in a process where a given item of data, or work product, is used. A programmer can sometimes do that himself for a small problem, but for a complex process, it would be a major optimization task which would consume time and introduce bugs. Therefore the ECL compiler performs a complete analysis of the program to ensure work is only done as often as required.
3) **Keeping all of the machines busy**. For those of us that remember FORTRAN, we are probably still stuck in the idea that data is read-in, processed and written out. That is a simple and intuitive processing model; which just about guarantees that two thirds of the machine resources are idle at any one time. ECL takes the opposite approach. While it allows processes to be specified intuitively, it will then re-arrange the processing schedule and group together tasks to ensure that all of the major machine components (disk, network, CPU, memory) are maxed out all the time. Of course by optimizing machine utilization you are also minimizing the latency of the process running.

How Enterprise Control Language Works

Another feature that gives ECL a distinctive and somewhat unusual flavor is that it was designed from the outset to run a huge organization for a long time. It did not evolve from an academic experiment; it was created as a medium through which hundreds of man years of data expertise could be expressed. The principle structure that allows this to happen is the module. A module is fully encapsulated; attributes only escape a module if they are explicitly exported. Even then attributes can only find their way into another module if they are explicitly imported. These language structures are then supported by the development tool which allows for multi-developer source control, repository searching and dependency checking.

A corollary of the design of ECL is that it allows and even encourages three different resource (programmer) deployment schemes:

1) **Lots of Simple Processing**: if your requirements are to do lots and lots of simple data processes then ECL can be used by almost anybody with some degree of technical proficiency. Many relatively complex data tasks are simply built into the language with a plethora of options; the chances are ECL will allow the simple process to occur with little or no effort.
2) **The 'Moon Landing' task**: ECL is all about abstraction and re-use. If a very complex task is to be undertaken by a smallish team of highly skilled people then ECL is a great way to allow a war-chest of best-practices to be built. Of course if the data is huge as well as the task being complex then all of the optimization features will be useful too!
3) **Lots of Complex Processing**: here the problem is that 'lots' and 'complex' tends to run counter to the resources we have available. ECL specializes in this field. The method is that the strongest developers work first extending the ECL language into the field in which the complex processing is required. Having performed these extensions then the complex processes are now wrapped in simple declarative words and to all intents and purposes the problem has now become one of lots of simple processing, against which a wider range of people can be deployed.

Learning ECL

Perhaps the final question is: 'how easy is ECL to learn?' In fact the answer is: it depends how hard it is for you to unlearn what you know. People coming to ECL with experience of a wide variety of different language types pick up ECL in a couple of days with joyful whooping noises. IBM 370 assembler programmers have come to ECL in a couple of weeks with sighs of relief. Some people that can recite the top 10 reasons for Java being the world's best programming language are still fighting the ECL compiler a couple of years later. The fact is that ECL has a small grammar and an extremely small set of operators. It has huge libraries but they are all hidden beneath the small grammar. The learning task is thus one of thinking declaratively rather than trying to "force the compiler into doing such and such."

Here is a simple example to return a list of the performers that have appeared most often in the movies database:

```
F0 := IMDB.File_actors;
CountActors := RECORD
F0.ActorName;
UNSIGNED C := COUNT(GROUP);
END;
MoviesIn := TABLE(F0,CountActors,ActorName);
OUTPUT(TOPN(MoviesIn,100,-C));
```

Comparing ECL to Other Solutions

It can be seen that ECL is very different. It is probably the most significant difference between the HPCC and Hadoop (which is Java based). When embraced, it allows data processes to run an order of magnitude faster than a Hadoop equivalent while also allowing the programmer to do an order (or two) of magnitude less work than a Hadoop coder. ECL is restricted to processes which are data centric but has language extension to allow highly specialized data processing tasks to be specified both concisely and precisely. The ECL optimizer then takes the ECL program and maximizes the execution performance against a given target cluster. Aspects of ECL are available to most technically capable people but optimum results require a willingness to fully embrace the declarative paradigm.

For more information:
Website: http://hpccsystems.com/
Email: info@hpccsystems.com
US inquiries: 1.877.316.9669
International inquiries: 1.678.694.2200

About HPCC Systems

HPCC Systems from LexisNexis® Risk Solutions offers a proven, data-intensive supercomputing platform designed for the enterprise to solve big data problems. As an alternative to Hadoop, HPCC Systems offers a consistent data-centric programming language, two processing platforms and a single architecture for efficient processing. Customers, such as financial institutions, insurance carriers, insurance companies, law enforcement agencies, federal government and other enterprise-class organizations leverage the HPCC Systems technology through LexisNexis® products and services. For more information, visit http://hpccsystems.com.

About LexisNexis Risk Solutions

LexisNexis® Risk Solutions (http://lexisnexis.com/risk/) is a leader in providing essential information that helps customers across all industries and government predict, assess and manage risk. Combining cutting-edge technology, unique data and advanced scoring analytics, Risk Solutions provides products and services that address evolving client needs in the risk sector while upholding the highest standards of security and privacy. LexisNexis Risk Solutions is headquartered in Alpharetta, Georgia, United States.

LexisNexis®