Welcome

HPCC Systems
http://hpccsystems.com

Database Week - NYC

Tackling Big Data with HPCC Systems, Hadoop & Pentaho BI Suite

Dr. Flavio Villanustre, VP Infrastructure & Products, LexisNexis & Head of HPCC Systems

Agenda

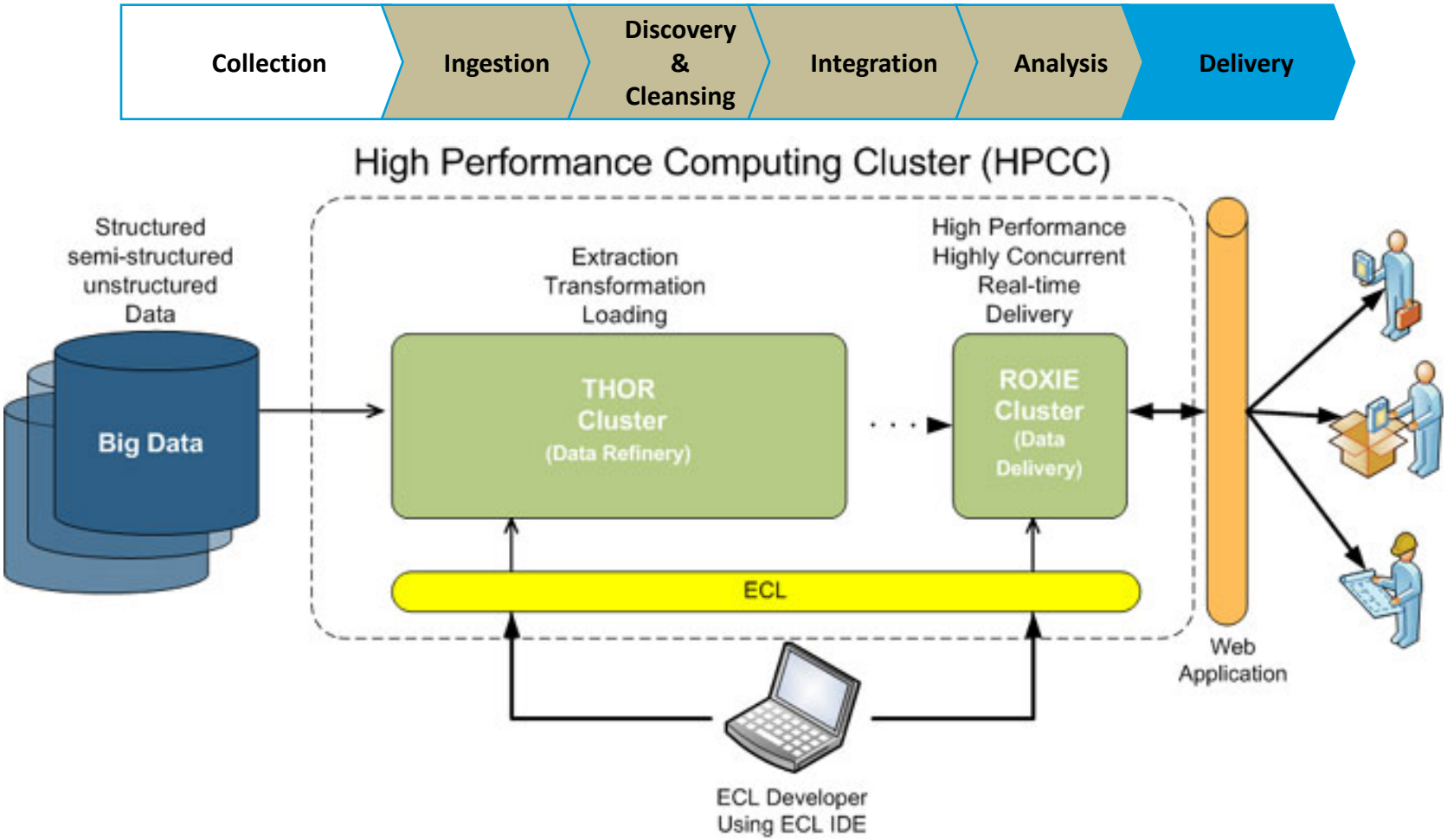
6:30-6:45pm:	Welcome / Power Networking
6:45-8:00pm:	Presentation & Demo
8:00-8:20pm:	Q&A / Open Discussion
8:20-8:30pm:	Kindle Fire giveaway / Close

Twitter event hashtag:

#hpccmeetup

hpccsystems.com

What is HPCC Systems?

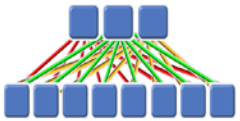


The Three main HPCC components

1 HPCC Data Refinery (Thor)

- Massively Parallel Extract Transform and Load (ETL) engine
  - Built from the ground up as a parallel data environment. Leverages inexpensive locally attached storage. Doesn't require a SAN infrastructure.
- Enables data integration on a scale not previously available:
  - Current LexisNexis person data build process generates 350 Billion intermediate results at peak
- Suitable for:
  - Massive joins/merges
  - Massive sorts & transformations
  - Programmable using ECL

2 HPCC Data Delivery Engine (Roxie)



- A massively parallel, high throughput, structured query response engine
- Ultra fast low latency and highly available due to its read-only nature.
- Allows indices to be built onto data for efficient multi-user retrieval of data
- Suitable for
  - Volumes of structured queries
  - Full text ranked Boolean search
- Programmable using ECL

3 Enterprise Control Language (ECL)

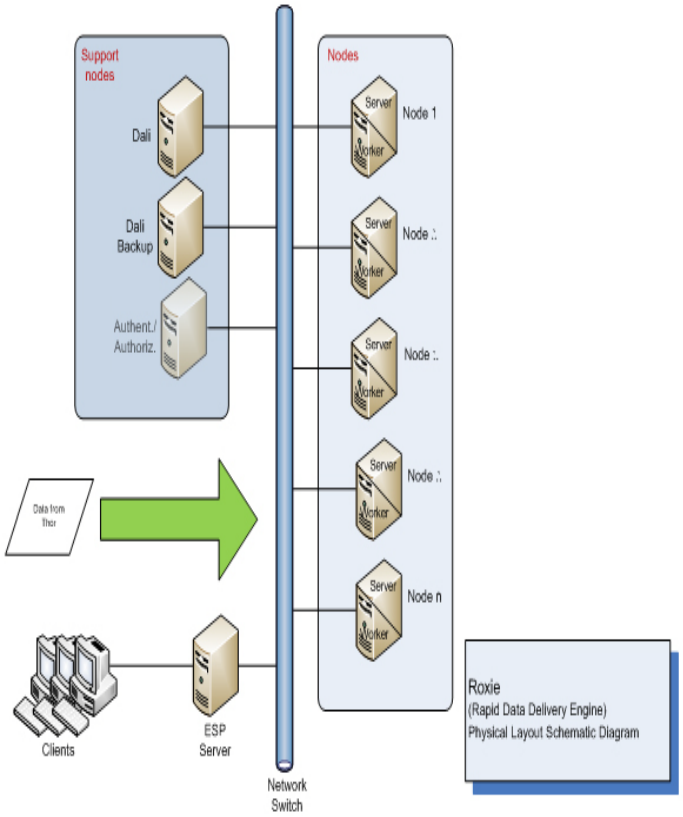
- An easy to use , data-centric programming language optimized for large-scale data management and query processing
- Highly efficient; Automatically distributes workload across all nodes.
- Automatic parallelization and synchronization of sequential algorithms for parallel and distributed processing
- Large library of efficient modules to handle common data manipulation tasks

Conclusion: End to End solution

- No need for any third party tools (Cassandra, GreenPlum, MongoDB, RDBMS, Oozie, Pig, etc.)

Roxie Delivery Engine

- **Low latency:** Data queries are typically completed in fractions of a second
- **Not a key-value store:** Unlike HBase, Cassandra and others, Roxie is not limited by the constrains of key-value data stores, allowing for complex queries, multi-key retrieval, fuzzy matching and real-time analytics
- **Highly available:** Roxie is designed to operate in critical environments, under the most rigorous service level requirements
- **Scalable:** Horizontal linear scalability provides room to accommodate for future data and performance growth
- **Highly concurrent:** In a typical environment, thousands of concurrent clients can be simultaneously executing transactions on the same Roxie system
- **Redundant:** A shared-nothing architecture with no single points of failure provides extreme fault tolerance
- **ECL inside:** One language to describe both, the data transformations in Thor and the data delivery strategies in Roxie
- **Consistent tools:** Thor and Roxie share the same exact set of tools, which provides consistency across the platform



The HPCC cluster computing paradigm and an efficient data-centric programming language are key factors in our company's success

Distributed Computing

Splits problems into pieces to be worked in parallel by commodity servers

+

Data-centric language (ECL)

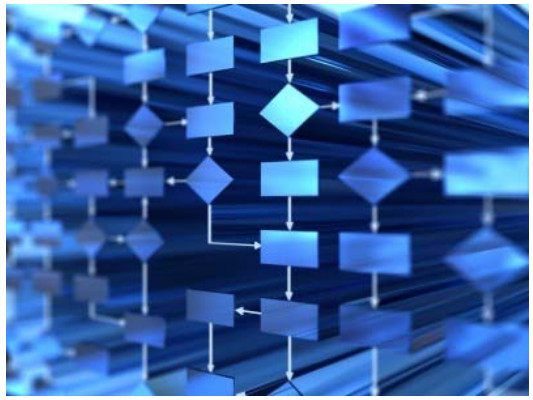
"Big Data" language brings the computing to the data

```
// Initialize output log
log_out_init := project(log_init,
                        transform(layout_logo
                        self := left,
                    self := []));

// Create error log
outerrorfile := join(log_seq,
                     log_out_init,
              left.linenum = right.linenum,
              transform(recordof(log_seq),
                        self := left),
              left only,
              hash);

// Denormalize key value pairs
outlogfile := sort(denormalize(distribute(log
                               sort(distribute(key
                  left.linenum = right.linenum,
                  transform(layout_logout,
                            self.keyvals := left
                                            row({ri
                        self := left),
```

=

Integrated Delivery System

Consistent approach across data ingestion, processing, and delivery systems



ECL: Enterprise Control Language

- **Declarative programming language:** Describe **what** needs to be done and **not how** to do it
- **Powerful:** Unlike Java, high level primitives as JOIN, TRANSFORM, PROJECT, SORT, DISTRIBUTE, MAP, etc. are available. Higher level code means fewer programmers & shortens time to delivery
- **Extensible:** As new attributes are defined, they become primitives that other programmers can use
- **Implicitly parallel:** Parallelism is built into the underlying platform. The programmer needs not be concerned with it
- **Maintainable:** A high level programming language, no side effects and attribute encapsulation provide for more succinct, reliable and easier to troubleshoot code
- **Complete:** Unlike Pig and Hive, ECL provides for a complete programming paradigm.
- **Homogeneous:** One language to express data algorithms across the entire HPCC platform, including data ETL and high speed data delivery.

```
// Initialize output log
log_out_init := project(log_init,
                        transform(layout_logo
                        self := left,
                  self := []));

// Create error log
outerrorfile := join(log_seq,
                     log_out_init,
           left.linenum = right.linenum,
           transform(recordof(log_seq),
                     self := left),
           left only,
           hash);

// Denormalize key value pairs
outlogfile := sort(denormalize(distribute(log_
                               sort(distribute(key
                left.linenum = right.linenum,
                transform(layout_logout,
                          self.keyvals := left
                                          row({ric
                     self := left),
```

The ECL programming paradigm

Paraphrasing Paul Graham (from the introduction to “On Lisp”):

In ECL, you don't just write your program down toward the language, you also build the language up toward your program. As you're writing a program you may think "I wish ECL had such-and-such an operator”. So you go and write it. Afterward you realize that using the new operator would simplify the design of another part of the program, and so on. Language and program evolve together. Like the border between two warring states, the boundary between language and program is drawn and redrawn, until eventually it comes to rest along the mountains and rivers, the natural frontiers of your problem. In the end your program will look as if the language had been designed for it. And when language and program fit one another well, you end up with code which is clear, small, and efficient.

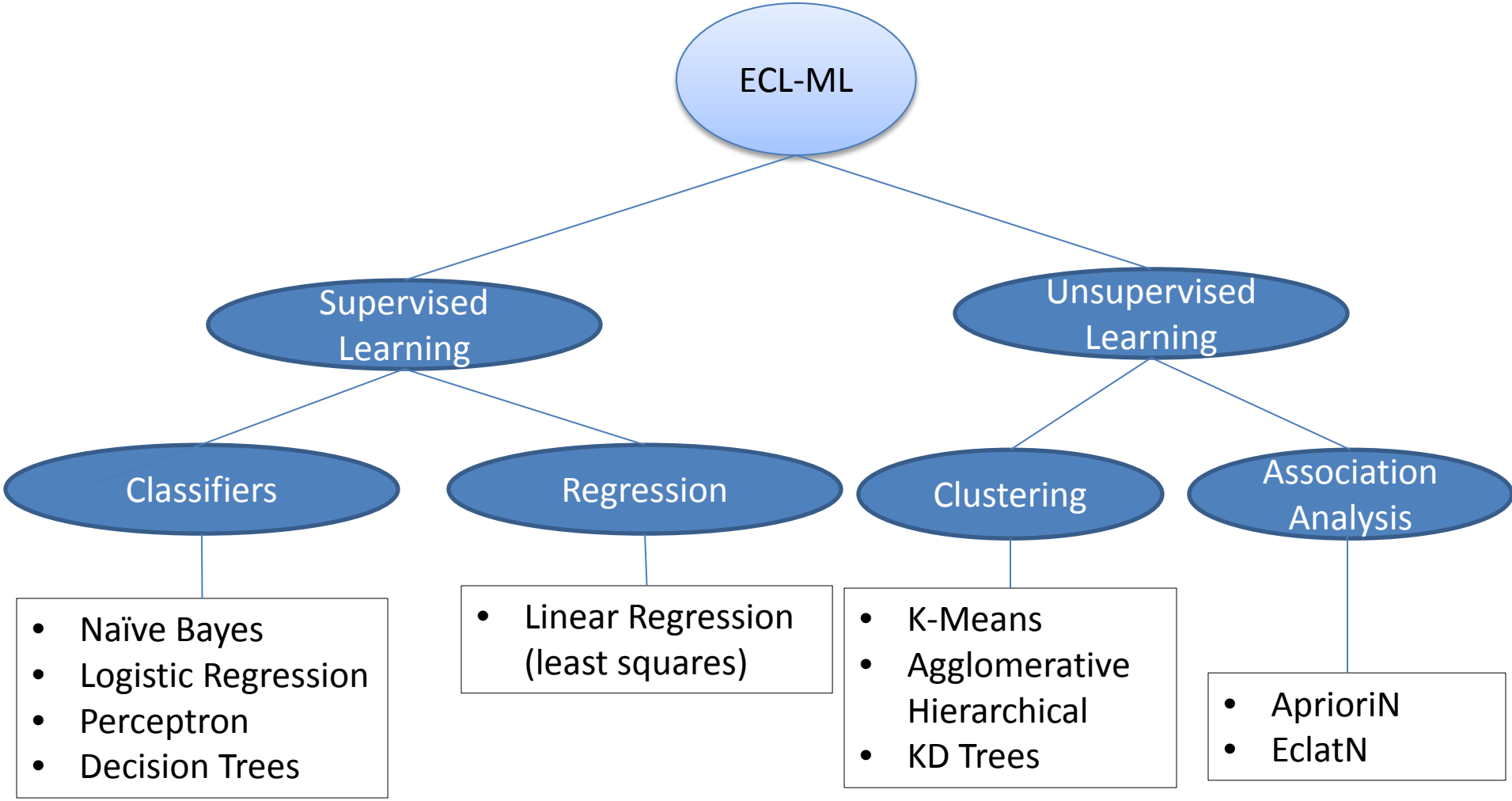
ECL for ETL

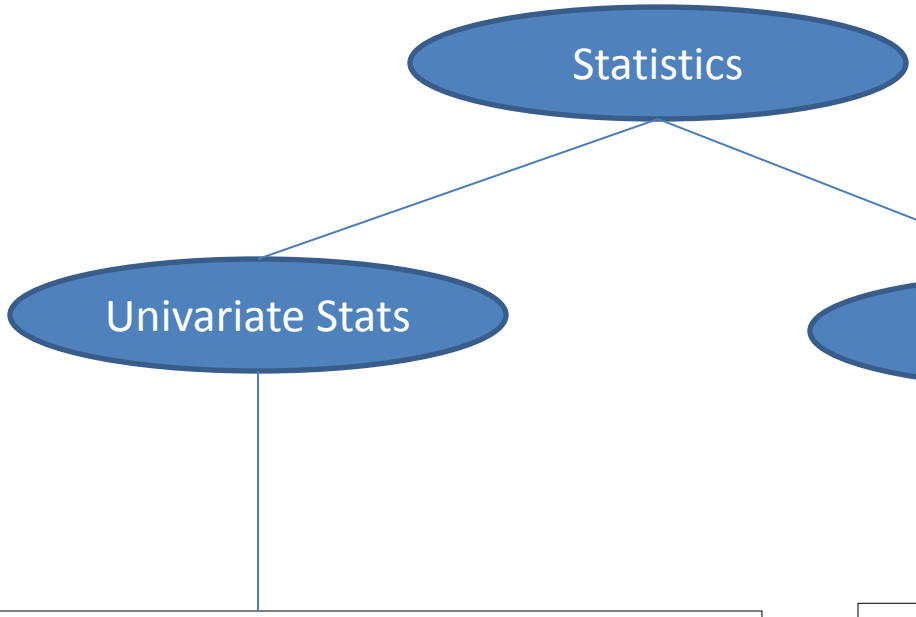
Basic Structure	PersonRec := RECORD
	  STRING50 firstName;
	  STRING50 lastName;
	  UNSIGNED1 age;
	END;

Transformations	PersonRec personTransform(PersonRec person) := TRANSFORM
	  SELF.upperFirstName := UPPER(person.firstName);
	  SELF := person;
	END;
	
	upperPersons := PROJECT(persons, personTransform(LEFT));
	OUTPUT(upperPersons);

Functions Used in context of Transformations	PROJECT, ROLLUP, JOIN, ITERATE, NORMALIZE, DENORMALIZE
All Functions	http://hpccsystems.com/community/docs/ecl-language-reference/html/built-in-functions-and-actions

ECL ML

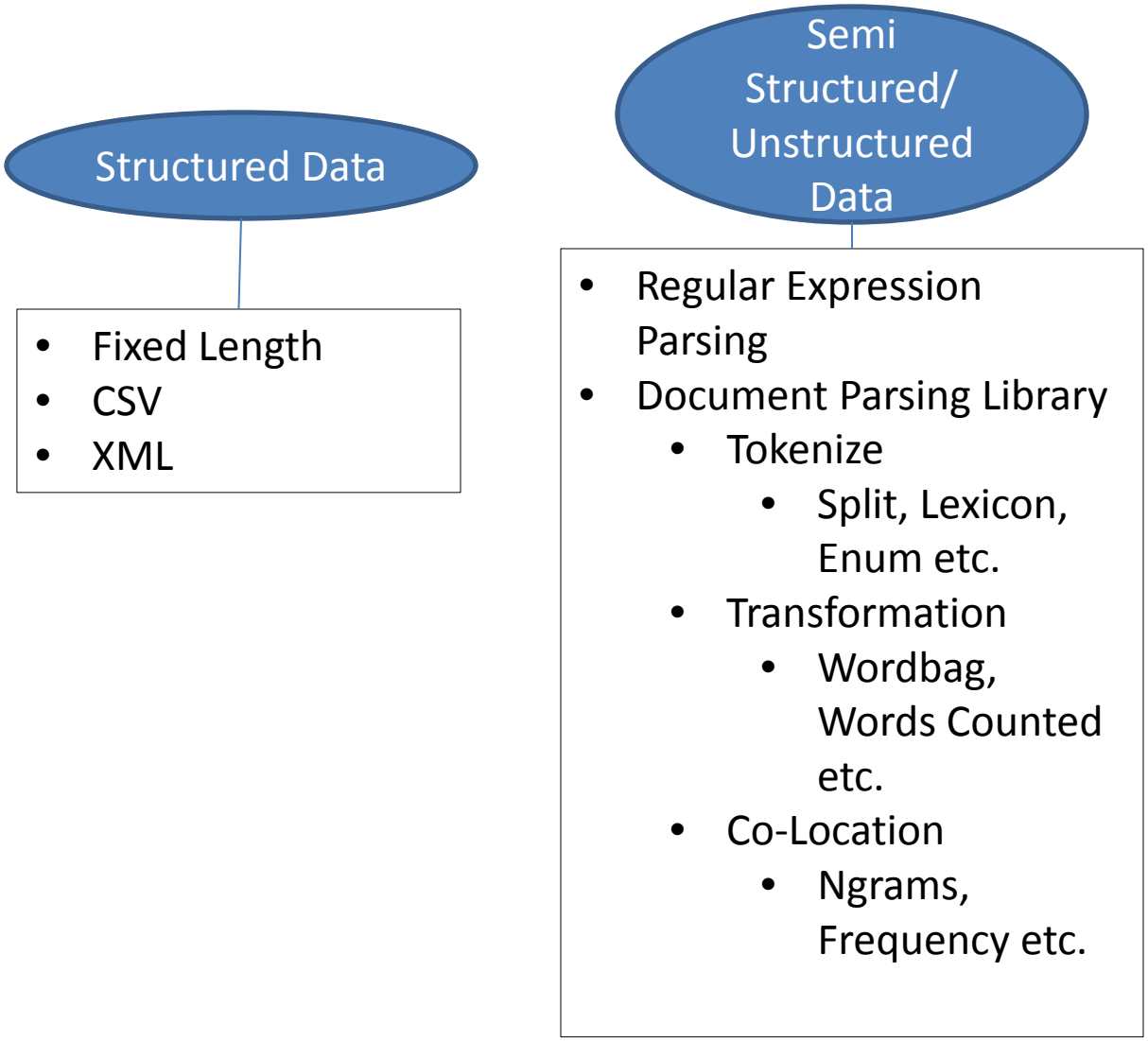

ECL-ML
Supervised Learning
Unsupervised Learning
Classifiers
Regression
Clustering
Association Analysis
• Naïve Bayes
• Logistic Regression
• Perceptron
• Decision Trees
• Linear Regression (least squares)
• K-Means
• Agglomerative Hierarchical
• KD Trees
• AprioriN
• EclatN



- Field Aggregates
- Mean, Variance, Std Deviation, max, min etc
- Medians, Modes, Cardinality, Buckets/Bucket Ranges, Simple Ranked, Ranked, NTiles/NTileRanges

- Pearson and Spearman
- Kendal's Tau

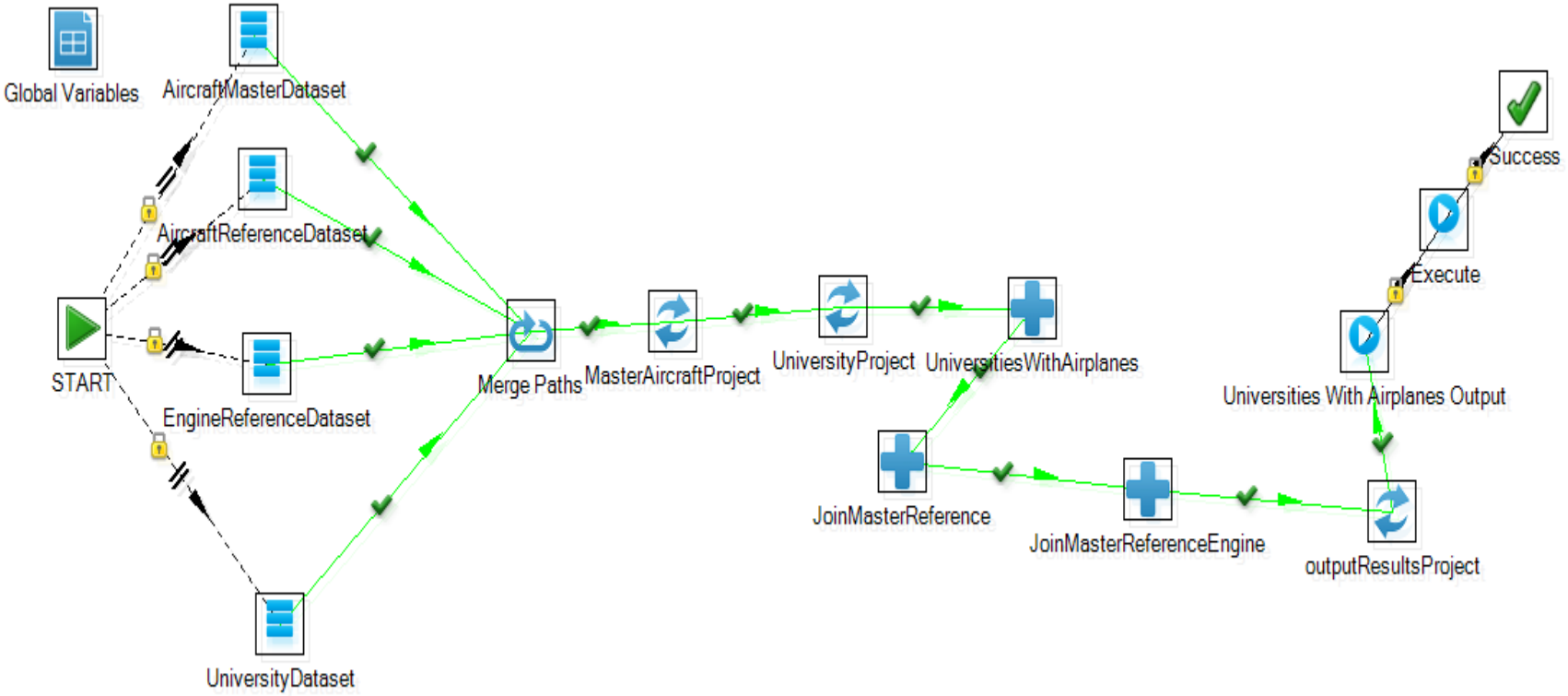
Parsing and Data Handling



Kettle – ECL General Functions

- SPRAY
- DESPRAY
- DATASET
- PROJECT
- JOIN
- ITERATE
- DEDUP
- TABLE
- DISTRIBUTE
- ROLLUP
- PARSE

Pentaho Spoon

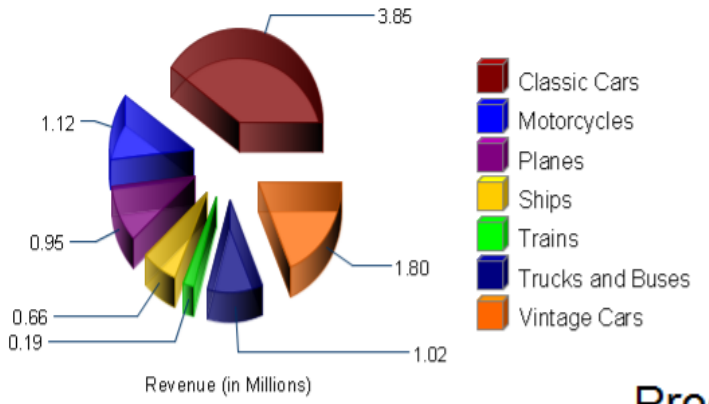


JDBC

- Standard support for SQL queries
- Filters
- Aggregation
- Joins
- Optimized by available Indexed fields

Reporting out of BIRT using HPCC JDBC (numbers are fictitious, of course!)

Product Line Revenue

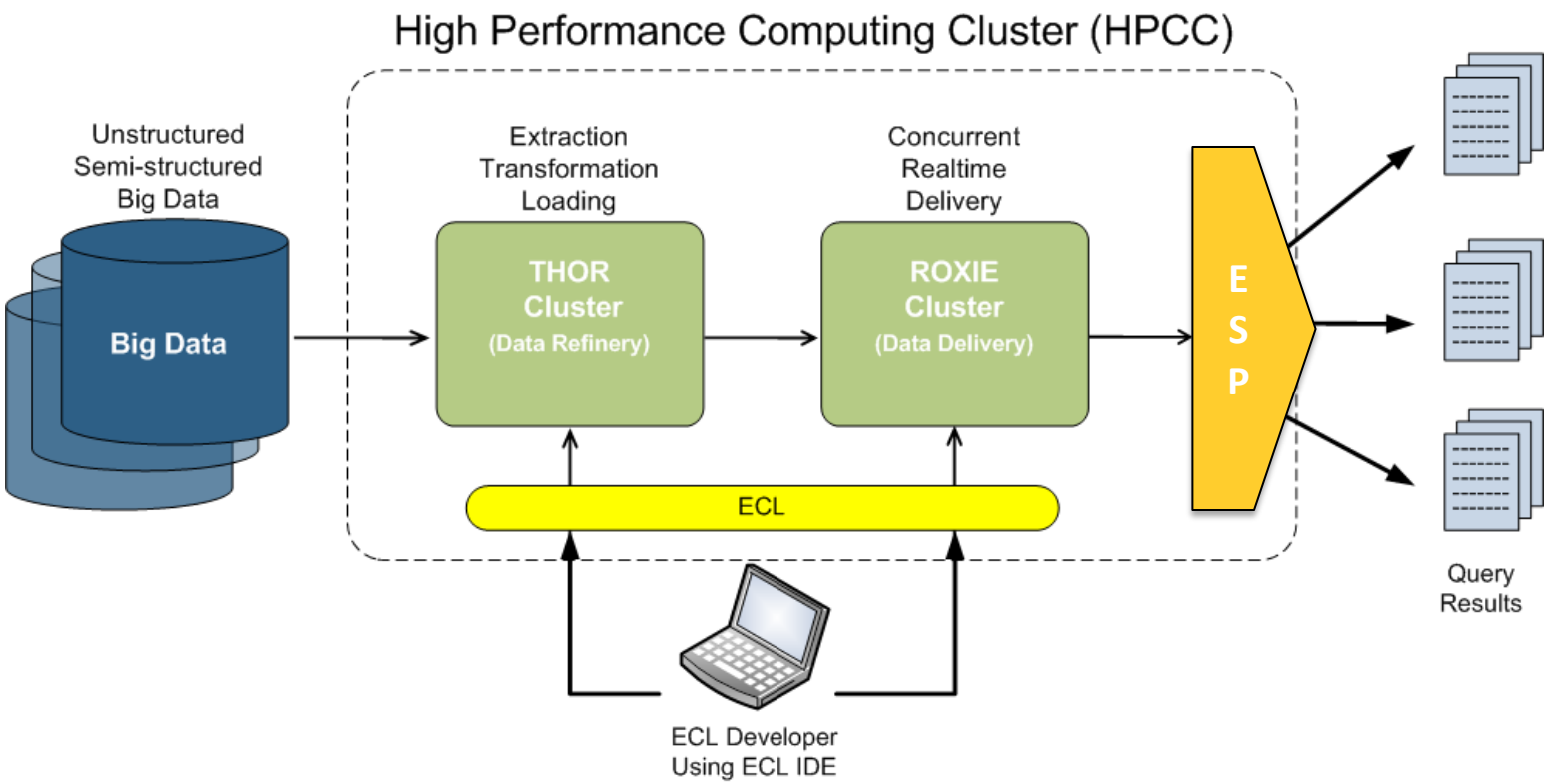


Product Revenue (Best Sellers First)

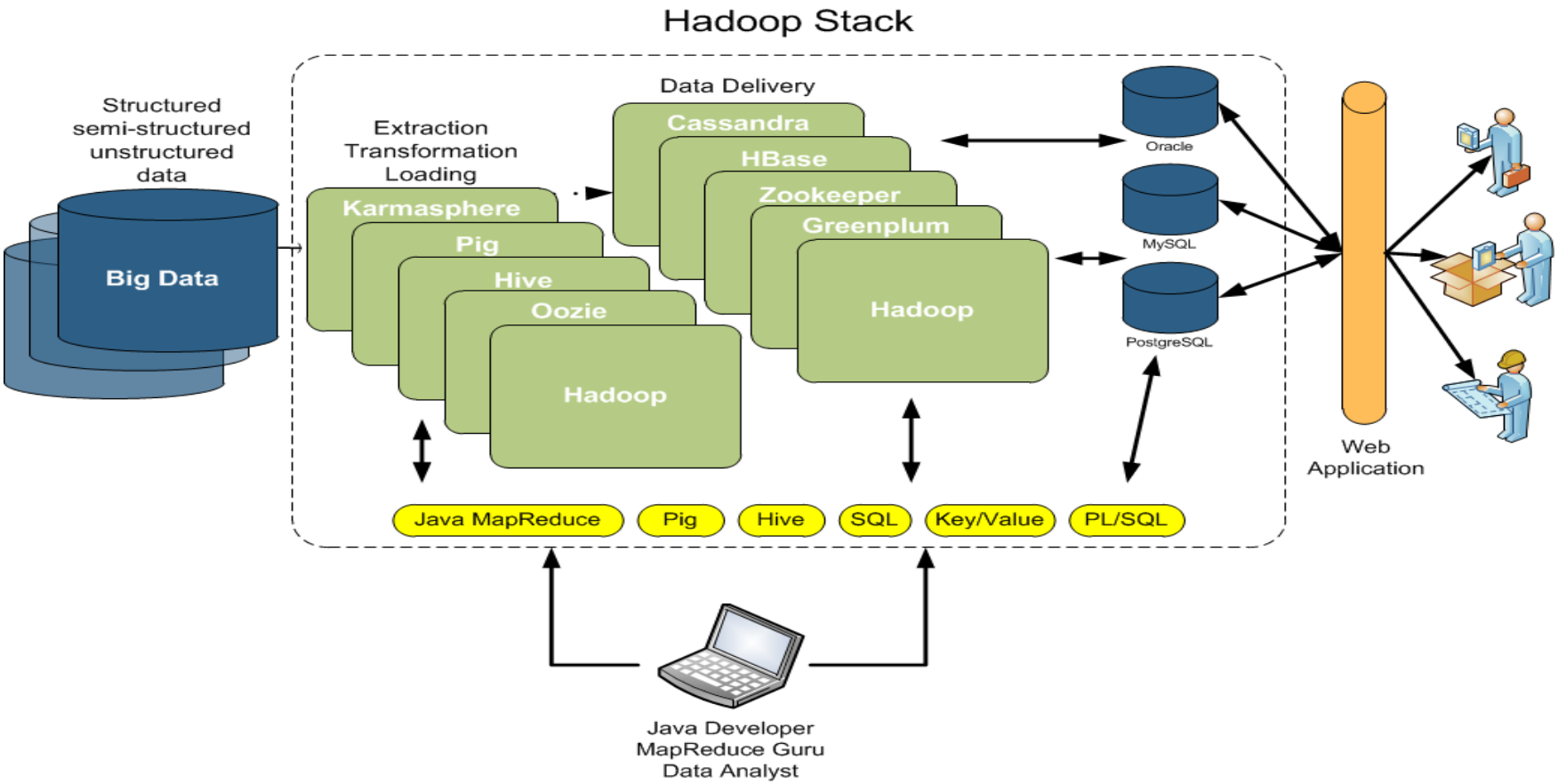
Product Name	Total Revenue
1992 Ferrari 360 Spider red	$276,839.98
2001 Ferrari Enzo	$190,755.86
1952 Alpine Renault 1300	$190,017.96
2003 Harley-Davidson Eagle Drag Bike	$170,686.00
1968 Ford Mustang	$161,531.48

COMPARISON BETWEEN THE HPCC PLATFORM AND HADOOP

The HPCC Systems platform architecture



Hadoop: Many platforms, fragmented, heterogeneous : Complex & Expensive



Data Model

HPCC supports a flexible data model: The data model is defined by the data analyst in the way it fits better the organizational skills, the data at hand and/or the process that needs to happen

Processing Model

HPCC supports a flexible dataflow oriented model: The complexities of parallel processing and distributed platform are abstracted behind high level data primitives

Programming Model

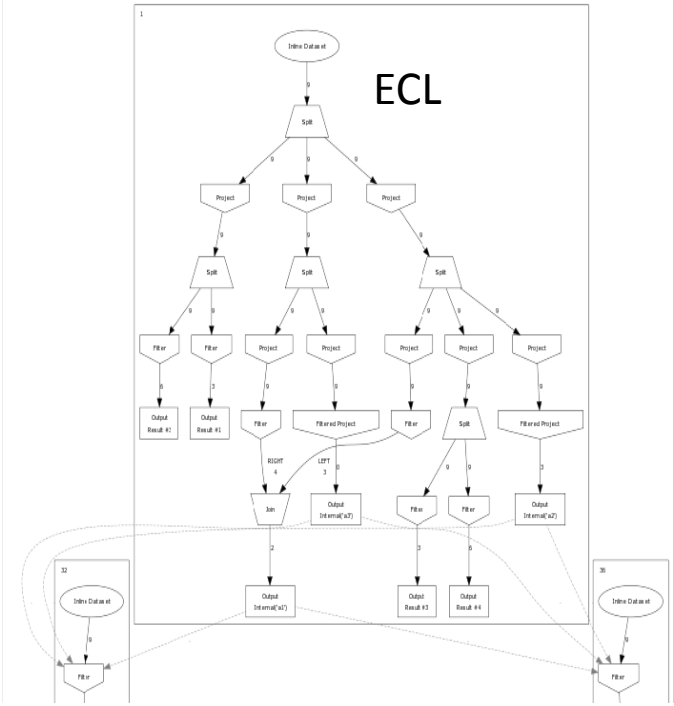
Data flows and data queries in HPCC are programmed in ECL: A complete high level declarative dataflow oriented language created for readability, extensibility and code/data reuse

Beyond MapReduce

- **Open Data Model:** Unlike Hadoop, the data model is defined by the user, and is not constrained by the limitations of a strict key-value paradigm

- **Simple:** Unlike Hadoop MapReduce, solutions to complex data problems can be expressed easily and directly in terms of high level ECL primitives. With Hadoop, creating MapReduce solutions to all but the most simple data problems can be a daunting task. Many of these complexities are eliminated by the HPCC programming model

- **Truly parallel:** Unlike Hadoop, nodes of a data graph can be processed in parallel as data seamlessly flows through them. In Hadoop MapReduce (Java, Pig, Hive, Cascading, etc.) almost every complex data transformation requires a series of MapReduce cycles; each of the phases for these cycles cannot be started until the previous phase has completed for every record, which contributes to the well-known “long tail problem” in Hadoop. HPCC effectively avoids this, which effectively results in higher and predictable performance.



- **Powerful optimizer:** The HPCC optimizer ensures that submitted ECL code is executed at the maximum possible speed for the underlying hardware. Advanced techniques such as lazy execution and code reordering are thoroughly utilized to maximize performance

Enterprise Ready

- **Batteries included:** All components are included in a consistent and homogeneous platform – a single configuration tool, a complete management system, seamless integration with existing enterprise monitoring systems and all the documentation needed to operate the environment is part of the package

- **Backed by over 10 years of experience:** The HPCC platform is the technology underpinning LexisNexis data offerings, serving multi-billion dollar critical 24/7 business environments with the strictest SLA's. In use by the US Government and Commercial settings for critical operations

- **Fewer moving parts:** Unlike Hadoop, HPCC is an integrated solution extending across the entire data lifecycle, from data ingest and data processing to data delivery. No third party tools are needed

- **Multiple data types:** Supported out of the box, including fixed and variable length delimited records and XML

"To boldly go where no open source data intensive platform has gone before"

H2H: HPCC/Hadoop data integration connector

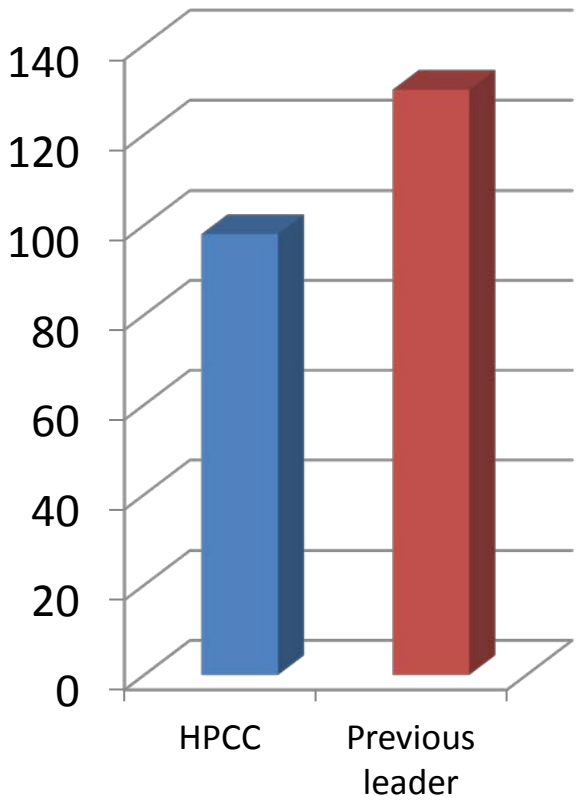
- Today
  - Read/write access to data stored in an HDFS filesystem
  - Seamless access from within ECL an ECL workunit: HDFS is just another type of data store
  - Perfect to use Roxie for real-time data delivery and complex analytics for data stored and processed in Hadoop
  - Parallel data access directly from/to the HDFS storage nodes
  - Beta just released! See: http://hpccsystems.com/h2h
- Future
  - Version 1.0 on its way!
- Even more future
  - Roxie for Hadoop: wizard based configurator to deploy and setup Roxie on top of Hadoop

One-Click Thor™ on AWS

- Launch an entire HPCC Thor system on AWS with a single click (well, maybe 2 or 3)
- Automatically:
  - Deploys support nodes
  - Configures the right parameters for the type of node/image
  - Implements ECLWatch for cluster management
  - Provides feedback on access URL's, configuration, etc.
  - Provides report (and logs) about the setup process
- Free! (you pay the AWS usage, of course) ☺
- Check out https://aws.hpccsystems.com for documentation and resources
- Join us June 26 at the NYC AWS meetup featuring the One-Click Thor solution

Terasort Benchmark results

Execution Time (seconds)



Productivity

```
// Perform global terasort
rec := record
      string10  key;
      string10  seq;
      string80  fill;
       end;
in := DATASET('nhtest::terasort1',rec,FLAT);
OUTPUT(SORT(in,key,UNSTABLE),,'nhtest::terasort1out',overwrite);
//End
```

3 Lines of ECL

```
  }
  abstract int findPartition(Text key);
  abstract void print(PrintStream strm) throws IOException;
  int getLevel() {
    return level;
  }
}

/**
 * An inner trie node that contains 256 children based on the next
 * character.
 */
static class InnerTrieNode extends TrieNode {
  private TrieNode[] child = new TrieNode[256];

  InnerTrieNode(int level) {
    super(level);
  }
  int findPartition(Text key) {
    int level = getLevel();
    if (key.getLength() <= level) {
      return child[0].findPartition(key);
    }
    return child[key.getBytes()[level] & 0xff].findPartition(key);
```

700+ Lines of Java MapReduce Code

Space/Cost

HPCC

Previous leader

Core Benefits

- **A simple technology stack results in more effective resource leveraging**
  - Fewer required skill sets means more flexibility in staffing projects
  - We no longer have silos of specialized expertise that can't be shared
- **A Single Data Store**
  - More efficient use of data and storage
  - Higher productivity across data analysts
  - Innovative products
  - Increased precision and recall
  - An online Data Warehouse
- **Better Analytics**
  - Faster scoring
  - Better classification and correlation
- **Speed**
  - Scales to extreme workloads quickly and easily
  - Increase speed of development leads to faster production/delivery
  - Improved developer productivity
- **Capacity**
  - Enables massive joins, merges, sorts, transformations

Appendix

The power of "what if…"

The power of the "what if…"

Imagine you had a platform designed for [Big Data] Data Analysts from the ground up?
Where organizations already invested in Data Analysts (BI) talent no longer have to worry about reinvesting in Java developers.
Or perhaps where new talent can be selected on the basis of data knowledge and creativity; rather than upon coding experience in a systems programming language. Java developers, by nature, are not necessarily BI or Big Data developers. And conversely, Data Analysts are not Java developers.

The HPCC Systems platform's ECL programming language was designed by data scientists for [Big Data] Data Analysts. It is made for people who think in terms of the "What if" rather than the "How".

Specifically **ECL is a declarative, data centric, distributed processing language for Big Data**. The declarative nature of the ECL language lets the analyst focus on finding the right question instead of the steps required to answer. Interactive iterative "what if" scenario running helps the data analyst pinpoint and formulate the correct questions faster.

ECL is to Big Data what SQL is to RDBMS. Good SQL developers will be able to quickly adapt to developing in ECL. Organizations investing in the HPCC Systems platform can reuse their existing BI staff to help solve the Big Data problems. No need for an additional layer of (Java) developers.

ECL is SQL on Steroids

	ECL	SQL
SELECT	persons	Select * from persons
FILTER	persons(firstName='Jim')	Select * from persons where firstName='Jim'
SORT	SORT(persons, firstName)	Select * from persons order by firstName
COUNT	COUNT(Person(firstName='TOM'))	Select COUNT(*) from Person where firstName='TOM'
GROUP	DEDUP(persons, firstName, ALL)	Select * from persons group by firstName
AGGREGATE	SUM(persons, age)	Select SUM(age) from persons
Cross Tab	TABLE(persons, {state; stateCount:= COUNT(GROUP);}, state)	Select persons.state, COUNT(*) from persons group by state
JOIN	JOIN(persons,state,LEFT.state=RIGHT.code)	Select * from persons,states where persons.state=states.code

Q&A
Thank You

Web: http://hpccsystems.com
Email : info@hpccsystems.com
Contact us: 877.316.9669