Data Intensive Computing

Programming models for the LexisNexis High Performance Computing Cluster

Topics

About LexisNexis Risk Solutions

About our Technology

About Data Intensive Computing

Programming Model

Adding a new Data Source

Creating a Linking Application

Conclusion

About LexisNexis®

- **World headquarters:** New York City
- **Parent company:** Reed Elsevier
- **Global reach:** customers in more than 100 countries
- **Employees:** 18,000 globally
- **Revenue:** $3.6 billion in 2008
- **Searchable online documents:** more than 5 billion
- **Information sources:** 40,000
- **Number of offices worldwide:** 110

All 50 states, 70 percent of local government and almost 80 percent of federal agencies use our services to make the world a safer place. In addition, we support 90 percent of the Fortune 500 companies and the leading non-profits in America.

LexisNexis®

Proven Industry Leader

LexisNexis® Risk Solutions is a leader for how information advances and protects people, industry and society:

- 20 billion public and proprietary records
- Coverage on more than 400 million individuals
- Coverage on more than 150 million business locations
- Patent-pending data matching and linking logic
- $1.5 billion in revenues in 2008
- Expanded services and solutions added with Reed Elsevier acquisition of ChoicePoint® (September, 2008)

LexisNexis Data Analytics Supercomputer

For more than thirty years, LexisNexis has been on the frontier of large-scale data management and analysis. Powered by high performance computing cluster technology (HPCC), the LexisNexis Data Analytics Supercomputer (DAS) was originally designed to:

- Solve the organization's internal data management and large-scale data analytics challenges.
- Deliver the speed and accuracy demanded by its expanding customer base.

Today, DAS powers all of the company's risk management solutions and helps customers solve large, complex data challenges such as national security issues.

DAS supports millions of transactions per day and performs 10 times faster than the next fastest system for large data analysis.

Intuitive Programming Language

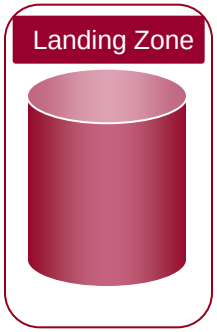
The core of DAS is the Enterprise Control Language (ECL). ECL programming efficiency is proven to be far greater than other approaches.

Declarative, non-procedural programming language optimized for large-scale data management and query processing

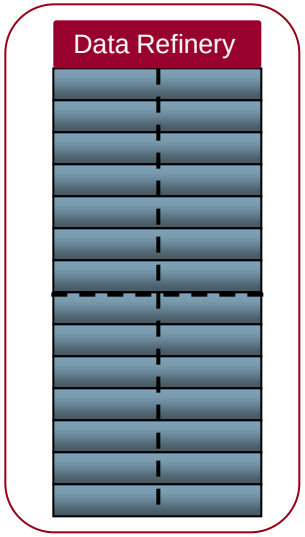
Automatically manages workload distribution across all nodes, including all aspects of the massive data joins, sorts and builds that truly differentiate DAS from other technologies in its ability to provide flexible data analysis on a massive scale.

Benefits programmers, who do not need to understand how to manage the parallel processing environment. As a result, users can express complex queries with less programming time and fewer lines of code than other conventional programming languages.

Hardware Architecture Components



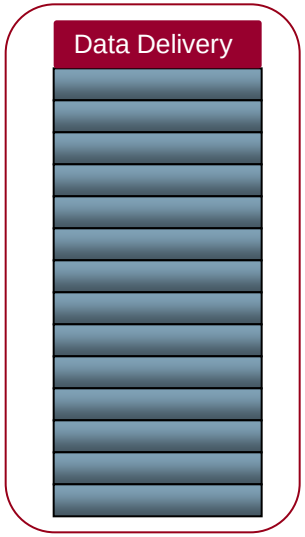
LZ: Disc Storage, sized as needed



Clusters sizes of 40-400 nodes

Unlimited number of clusters

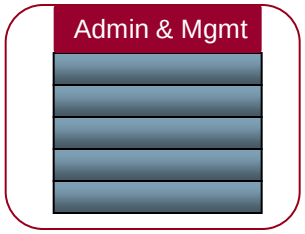
Multi-THOR: overlapping virtualized clusters running on same hardware



Clusters sizes of 1-100 nodes

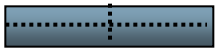
Unlimited number of clusters.

Typically 2-3 clusters load-balanced for 24x7 uptime



Typically 10 admin nodes for a system.

1 additional for each THOR cluster

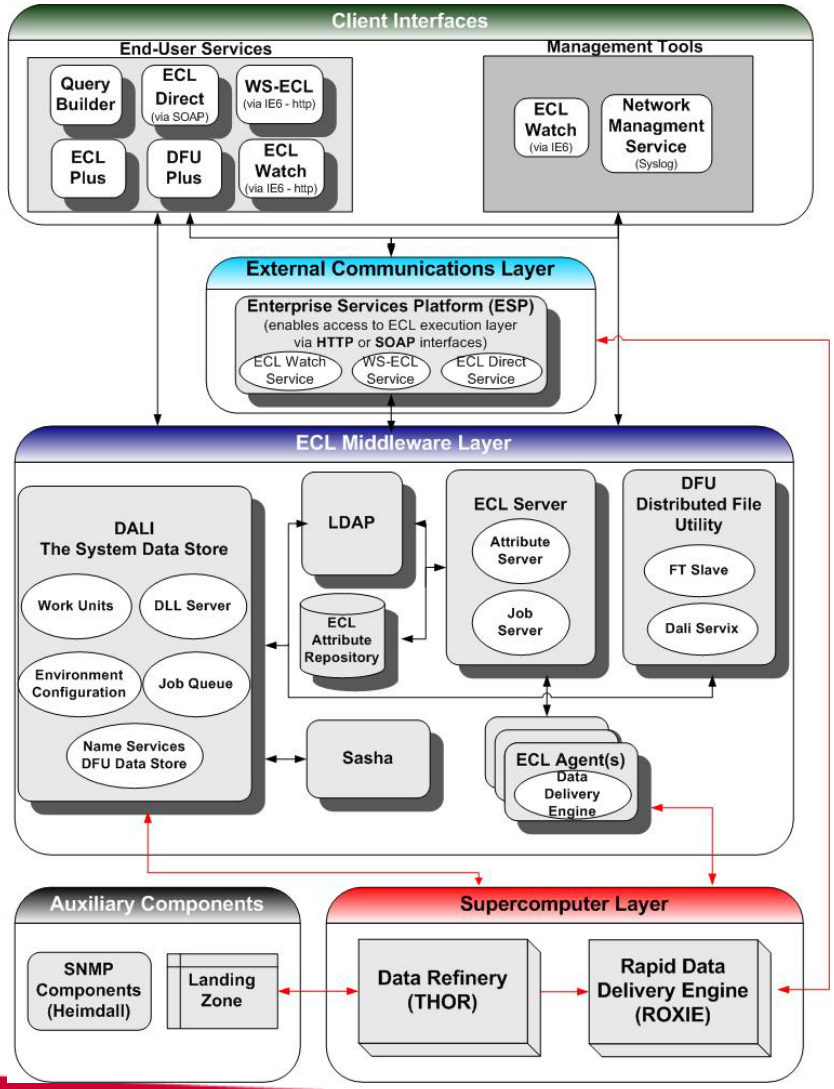


2U config contains 4 nodes and redundant power supplies. Each node has:

- Intel Xeon, Quad core, hyper-threaded processor (2.27 GHz), model # E5520
- **2TB** of available storage
- RAID 5 support
- 3 drives (2TB each drive). This includes redundancy due to RAID and mirroring
- 12GB RAM
- 8MB L2 cache

Force 10 Non-Blocking Switch, sized accordingly

Software Architecture Components



Data Intensive Computing

- **Traditionally supercomputing was focused on compute-intense problems such as weather forecasting and crash simulations. Compute-intense applications also create data that needs to be managed.**

- **Now a new type of supercomputing has emerged – data intensive supercomputing clusters -- to focus on data-intense problems driven by the volume of data generated by digital technology. Everybody has tons of data, but the question is how do you optimize it.**

- **Relational database solutions failed to provide adequate performance**
  - **Expensive**
  - **Slow to process data**
  - **Inflexible**
  - **Internal Research concluded that "shrink-wrapped" software solutions cannot address problem**

Data Intensive Computing Examples

Data Fusion and Linking

- Thousands of disparate sources, few or no equality match fields
- Mostly fuzzy or probabilistic match criteria

Data Mining

- Association mining and rule discovery
- Clique discovery

Risk Modeling and Analysis

Programming Model

There are Attributes and Actions

- Attributes declare or defines “what is to be done”
- Actions trigger work

Attributes are inherently re-usable

Attributes are very much like functions

Programming Model (continued)

Example of attributes and actions:

```
FirewallLogRecord := RECORD                          // Define how data is
    stored
    UNSIGNED4     sourceIP;
    UNSIGNED4     destinationIP;
    UNSIGNED2     sourcePort;
    UNSIGNED2     destinationPort;
    UNSIGNED4     bytesTransferred;
    UNSIGNED8     dateTimeStamp;
END;
dsFirewallLog := DATASET('My::Firewall::data', FirewallLogRecord, FLAT);
COUNT(dsFirewallLog);              // An action, perform a count
topTransferRecords(c=10) := TOPN(dsFirewallLog, c, bytesTransferred);
OUTPUT(topTransferRecords(100));  // An action, writes to top 100
```

Programming Model (continued)

Good definitions are re-used

```
topSourceDestination := topTransferRecords(1);
selectedFirewallRecords
   := JOIN(dsFirewallLog, topSourceDestination,
               LEFT.sourceIP=RIGHT.sourceIP
               AND LEFT.destinationIP=RIGHT.destinationIP,
               TRANSFORM(FirewallLogRecord, SELF:=LEFT),
               LOOKUP);
OUTPUT(selectedFirewallRecords);
   // list all entries associated with the high transfer
   // Note definitions are independent wrt data nodes
```

Programming Model (continued)

Note that a sequence of records is a first class object

Iteration through a sequence of records is implied

In our example, the record distribution was opaque

Adding a new Data Source

- Data can be XML, Delimited Values (e.g., TSV, CSV), Fixed length records
- Encoding can be Unicode (UTF-8, UTF-16, UTF-32), ASCII, EBCDIC, or any single byte encoding
- Standard data types of binary, floating point, packed decimal, character strings, and binary strings

Adding a new Data Source (continued)

Steps:

- Copy initial data set or data sets to the landing zone and spray the data
- Create the attribute for the record definition
- Create the attribute for the dataset
- Create the record structure or structures for the normal form of the data if desired
- Create the attribute that provides the normal form data

Building a Linking Application

- Analysis of data once sprayed
  - Field value ranges and uniques
  - Detect dirty data
  - Field level cleaning and encoding
- Review field semantics for linking
  - Name of vessel versus name of person
- Normalize encoding, if possible
  - Non-equality matching

Building a Linking Application (continued)

Use of Multiple Blocking
- Multiple joins with various join criteria

Incorporate statistical data
- Weight of the field based upon specificity
- Weight of the value based upon frequency

Score the matches, and summarize

Record the linkages

Building a Linking Application

Examine the boundary for errors, and adjust weights or add dimensions

Examine data and work distribution at the nodes, looking for skew

- May need to perform explicit data distribution of naïve approach has too much skew

Add inquiry or analysis applications

Add automation for ongoing data loads

Conclusion

Shared nothing cluster of commodity hardware provides an inexpensive hardware platform

Declarative nature of ECL provides means for automatic parallelization

Tuning applications may involve tuning the data distribution