HPCC Systems®

Rapid Data Delivery Engine Reference

Boca Raton Documentation Team

LexisNexis® Risk Solutions

Rapid Data Delivery Engine Reference

Boca Raton Documentation Team

We welcome your comments and feedback about this document via email to <docfeedback@hpccsystems.com> Please include **Documentation Feedback** in the subject line and reference the document name, page numbers, and current Version Number in the text of the message.

2013 Version 3.10.8.2

Introduction 4
Roxie Overview 5
Payload INDEXes 6
Roxie Superfiles 7
How Roxie Works 8
Roxie Data Backup 12
Developing Roxie Queries 14
Development Path 14
Methods to Submit Jobs to a Roxie Cluster 15
Managing Queries 18
Adding a roxie query to the Query Set 19
Viewing Query Sets using ECL Watch 20
Using WsECL to run a roxie query 21
Packages and Packagemaps 22
Example 23
Updating Data 24
Package File Syntax 25
Working with Packages using the ecl command line tool 26
Deploying Data to a Cluster using DFU 35
DFU Copy 36
Remote Copy 38
Capacity Planning for Roxie Clusters 39
Capacity Planning 39
PreFlight and Roxie Metrics 41
ECL Watch / Preflight 42
Roxie Queries and Roxie Files in ECL Watch 46
Roxie Metrics 51

Introduction

The Roxie engine - also known as the Rapid Data Delivery Engine or RDDE - uses a combination of technologies and techniques that produce extremely fast throughput for queries on indexed data housed in a dedicated High Performance Computing Cluster (HPCC).

Roxie queries can exceed several thousand a second, compared to Thor queries which tend to take from a few seconds to a few minutes each (from end to end) depending on the complexity of the query.

To fully understand this concept, it is best to look at the purpose for which each of these processes is designed:

- The Thor platform is designed to perform operations on every record (or most records) in a massive dataset.
- Queries can be run on the hThor platform if they can quickly pinpoint small sets of records within the data.
- Roxie queries are typically used to quickly pinpoint small sets of records over and over again.

If you think of all your data as an ocean, Thor would be used to perform operations on the entire ocean.

An hThor query might be used to find a single fish within that sea of data.

The query would be deployed to a Roxie cluster to be used to find hundreds or thousands of individual fish-one after another.

Roxie queries are deployed to a Roxie cluster, which pre-loads all queries into memory and prepares them to be ready to execute as soon as a query is received.

Queries are sent to Roxie via XML, SOAP, or JSON, and the results are returned in the same format. A client can communicate directly with the Roxie cluster by opening a socket to one of the servers in the cluster, or it can communicate via an ESP service such as WsECL.

Roxie Overview

There are typically four aspects to using Roxie:

- Creating indexes on datasets
- Using indexes in queries
- Compiling and Deploying queries to the Roxie Cluster
- Providing access to those queries to client-facing interfaces via SOAP or HTTP.

When to Use Indexes

The Thor platform is designed to perform operations quickly on massive datasets without indexes, where the entire dataset (or almost all of it) is to be used. However, if only a few records are needed, an index can access them more efficiently. In the ECL language, an index behaves just like a dataset that happens to be able to implement certain functions (typically filter and count functions) much more quickly than a standard flat file or CSV dataset can.

Payload INDEXes

In conventional database systems, an index is used together with a data file to locate records in that data file. You can do the same in ECL by storing file positions in the index and using them in a FETCH function to retrieve the corresponding data rows from the original file.

However, because you can store whatever fields you wish in an index, it is more common in Roxie queries to design indexes that store both the search fields and the information you want to look up. This eliminates an extra disk read for the FETCH. Since indexes are compressed, this can also save disk space if the original data file does not need to be stored on the Roxie cluster.

Any field in an index that does not need to be searched for can be specified as being in the "payload" - such fields are stored only at the leaf nodes of the index tree which can save some space and performance in the lookups. The fields in the payload may simply be additional fields from the base dataset, but they may also contain the result of some preliminary computation (computed fields).

Roxie Superfiles

A superfile used in Roxie may not contain more than a single sub-file (a superkey, however, may contain multiple payload indexes). This restriction makes using superfiles in Roxie just an exercise in file redirection. By writing queries that use superfiles (even though they contain only a single sub-file) you have the advantage of being able to update your Roxie by simply deploying new data without needing to re-compile the queries that use that data simply because the sub-file name changed. This saves compilation time, and in a production environment (which a Roxie typically is) where a data file is used by many queries, this savings can be a significant.

See the *ECL Programmer's Guide* for more details.

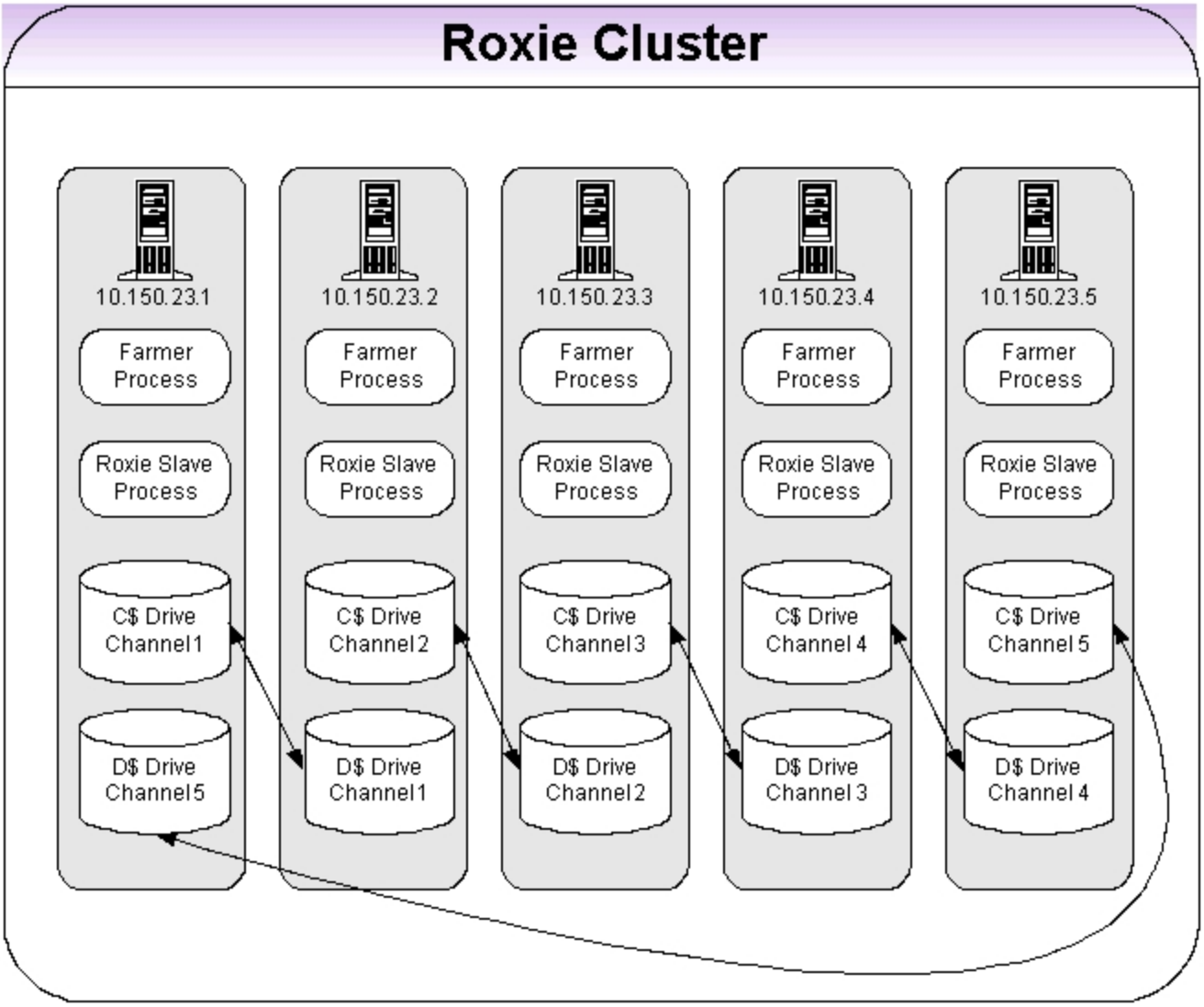
How Roxie Works

Roxie clusters consist of a number of machines connected together to function as a single entity. ECL source code for one or more queries is compiled and published to the cluster. Once published, queries can process data rapidly.

Each machine in the cluster acts in two distinct roles - these roles execute in the same process and share a lot of code - including the compiled query code - but can be thought of as logically distinct.

The **Server** process accepts incoming queries from a client, evaluates the ECL code of the query given the input that is provided in the client query, and returns the result to the client. When the Server evaluates an ECL function that requires data from disk, it determines the slave node or nodes that may have the appropriate data, and sends a request to those slave nodes to retrieve any matching data. Results from multiple slave nodes are collated and become the input for further ECL functions evaluated on the Server node. Typically, requesting applications use some form of load balancing to distribute requests evenly to the available Servers.

The **Slave** process accepts requests only from other Server nodes in the cluster. These requests correspond to a single ECL function such as a filtered index read or a disk fetch. Results are sent back to the Server that originally requested them. In order to balance performance and manage hardware failures, slaves receive the requests over multicast, and there will typically be at least two slave nodes that will receive each request from a server. The slave nodes communicate with each other to avoid duplicating effort, so that whichever slave is first able to handle the request tells the others not to bother. Each node in a cluster typically handles Slave requests on two or more multicast channels, usually one channel per disk drive. If any Slave node is not responding, the requests on that channel are handled by the other peer Slave nodes responsible for that channel.



This example shows a 5-node Roxie Cluster with each node configured to be both a **Server** and a **Slave**.

Queries that have been compiled with the target platform specified as a Roxie cluster may be published to a QuerySet using EclWatch.

Each Roxie cluster loads queries from one or more QuerySet lists.

When a query is added to the QuerySet that a Roxie is watching, Roxie will preload the query .so (or .DLL) and prepare the execution context as far as it can, so that it is ready to execute incoming requests to execute that query as soon as they arrive. This may includes loading the .so (or .DLL), resolving file references, and opening files (if there are sufficient file handles available), preloading data into memory if requested, and evaluating ECL code in the query that has been marked as : ONCE.

Depending on the configuration, Roxie may read data remotely from a Thor cluster where it was prepared, or if preferred, it may be copied to the Roxie for local access.

Typically a development system might refer to data in situ on the Thor cluster, while a production system may prefer the performance benefits of copying data locally to the Roxie.

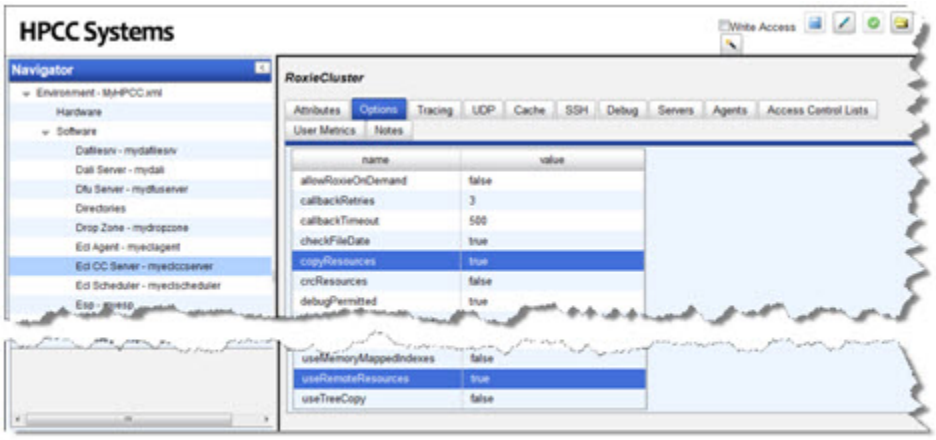
Roxie can read data remotely while it is being copied and switch to the local copy once the copy operation is complete. This provides the benefit of quick availability because the query can be active before the data is copied, while still taking advantage of the performance benefits of local data.

Queries and Data

Data files and index files referenced by a Roxie query's ECL code are made available in one of four ways, depending on the configuration of the Roxie cluster.

There are two settings in the Roxie configuration that control where Roxie looks for data and index files:

copyResources	Copies necessary data and key files from the current location when the query is published.
useRemoteResources	Instructs Roxie to look for data and key files in the current location after the query is published.



These options may appear to be mutually exclusive, but the chart below shows what each of the four possible combination means.

copyResources	T	T	F	F
useRemoteResources	T	F	T	F
	Directs the Roxie cluster to use the remote instance of the data until it can copy the data locally. This allows a query to be available immediately while the data is copied.	Directs the Roxie cluster to copy the data locally. The query cannot be executed until the data is copied. This ensures optimum performance after the data is copied.	Directs the Roxie cluster to load the data from a remote location. The query can be executed immediately, but performance is limited by network bandwidth. This allows queries to run without using any Roxie disk space, but reduces its throughput capabilities.	Will use data and indexes already loaded (placed on the Roxie cluster using DFU) but will not copy or read remote data.

When **copyResources** is enabled, data files are copied from source locations to slave nodes. Index files are also copied, with an extra file part that is a “key-of-keys” or a metakey.

A copy of the metakey is always stored on each Farmer and in most cases is loaded into memory at startup to increase performance.

Command Line support

The ecl command line tool offers a means of copying queries from one Roxie to another. Typically, this means data and index files would be copied or accessed remotely following Roxie’s configuration settings.

The ecl command line tool provides an option to instruct Roxie to not copy files at the time of the query copy. This allows you to quickly copy a query without copying files.

```
ecl queries copy --no-files
```

This specifies to NOT copy files referenced by the query being copied. The query cannot run until data is made available.

Roxie Data Backup

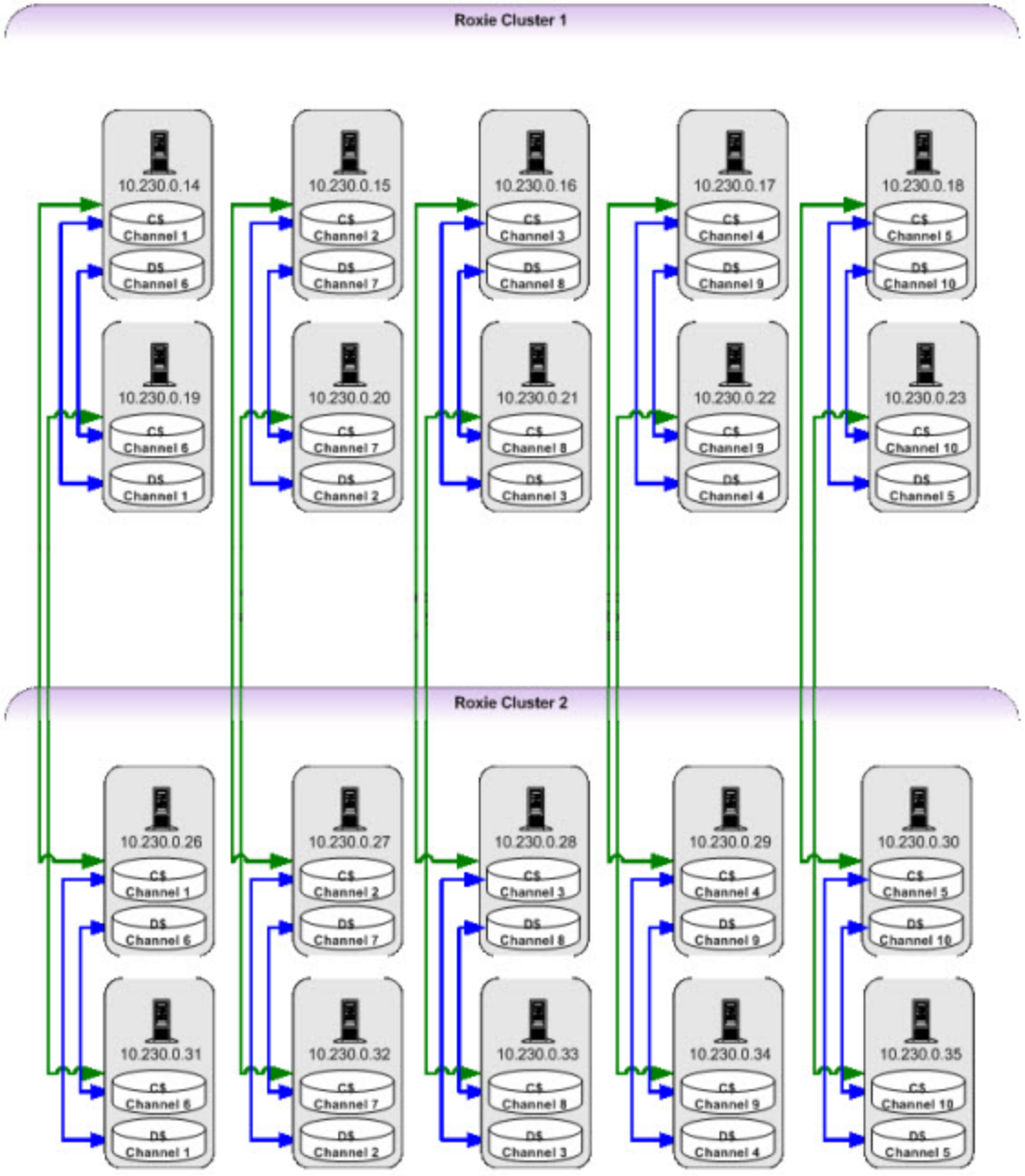
Roxie data is protected by three forms of redundancy:

- Original Source Data File Retention: When a query is deployed, the data is typically copied from a Thor cluster's hard drives. Therefore, the Thor data can serve as backup, provided it is not removed or altered on Thor. Thor data is typically retained for a period of time sufficient to serve as a backup copy.

- Peer-Node Redundancy: Each Slave node typically has one or more peer nodes within its cluster. Each peer stores a copy of data files it will read.

- Sibling Cluster Redundancy: Although not required, Roxie deployments may run multiple identically-configured Roxie clusters. When two clusters are deployed for Production each node has an identical twin in terms of data and queries stored on the node in the other cluster.

This configuration provides multiple redundant copies of data files. In this example, there are six copies of each file at any given time; eliminating the need to use traditional backup procedures for Roxie data files.



Developing Roxie Queries

Development Path

1. Determine the need.
2. Evaluate data and determine fields to index.
3. Build Index(es).
4. Create a hThor query.
5. Test and fine-tune the query (using hThor).
6. Publish the Query to a Roxie cluster.
7. Test and certify (compare results to expected results).

Note: These steps are explained in detail in the HPCC Data Tutorial and the Programmer's Guide.

Methods to Submit Jobs to a Roxie Cluster

After a query is compiled and deployed, there are several methods to submit jobs that use the query. While the most common usage is via custom applications using the XML or SOAP interface, the other methods have valid uses, too.

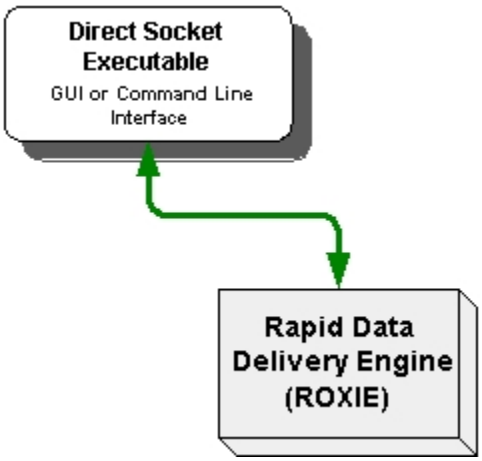
A direct socket connection can communicate directly with the Roxie cluster, eliminating all other intermediate components. This provides a means of certifying the Roxie cluster, its configuration, the deployment of the query, and the query itself.

SOAPCALL allows a Thor query to make calls to a Roxie query (See the *ECL Language Reference* for details). This provides the capability of combining Roxie results with other data processing tasks performed during ETL.

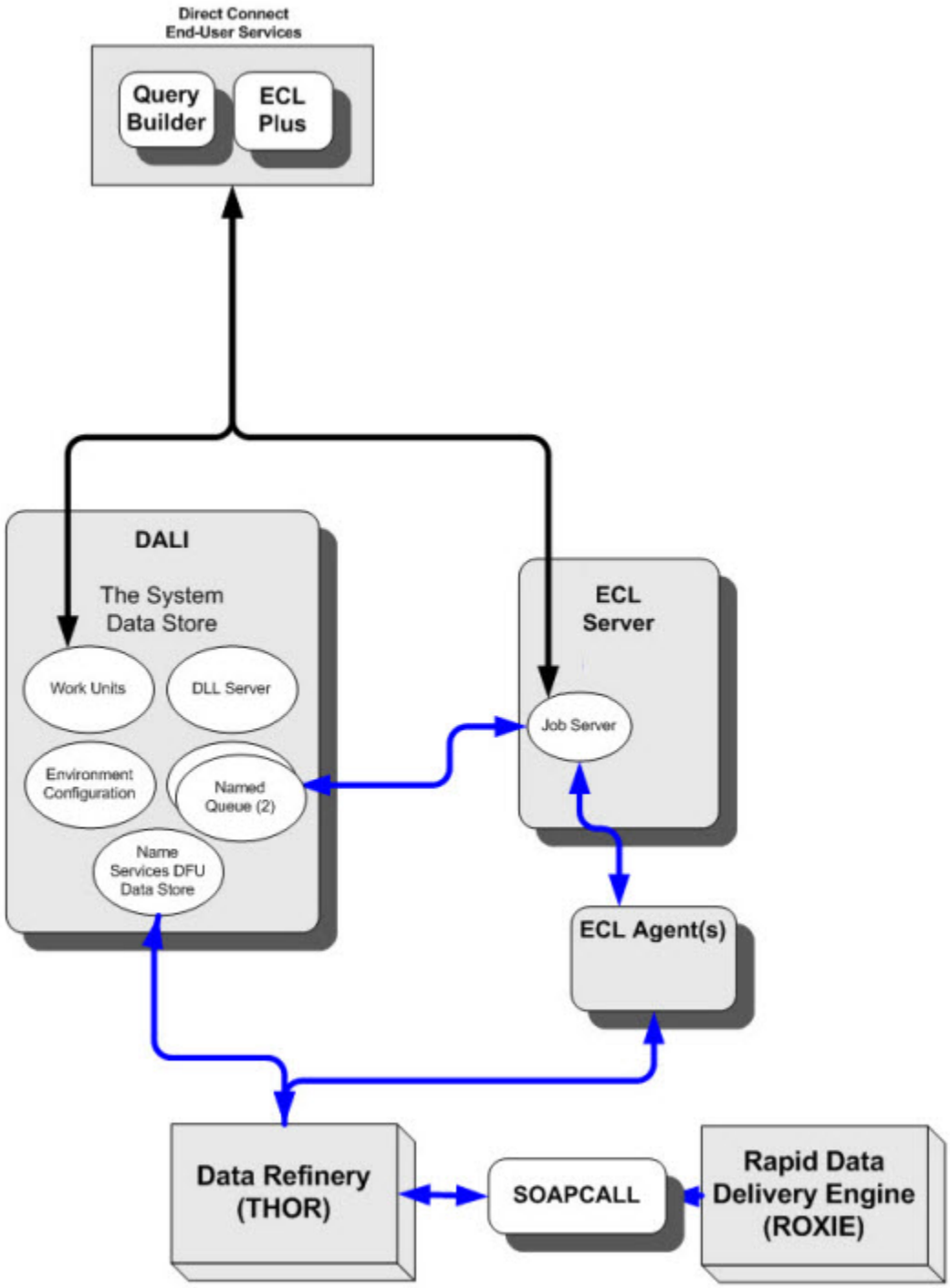
Conducting queries via an ESP service using HTTP or HTTPS allows access to queries directly from a browser. Web-based access allows you to provide easy access to anyone you wish. Using HTTPS, you can ensure data security using Secure Socket Layer (SSL) encryption. This ensures all data is encrypted as it travels across a network or the Internet. In addition, LDAP authentication is available to restrict access to a set of users.

Custom applications using SOAP provides the most flexibility and functionality. The application development process is simplified by Enterprise Services Platform's automatic Web Services Definition Language (WSDL) generation. Many development tools (such as, Microsoft's .NET Studio or NetBeans JAVA) include a tool to generate code to create proxy stubs from a WSDL document. This simplifies the development process and ensures exposure of all necessary methods and properties.

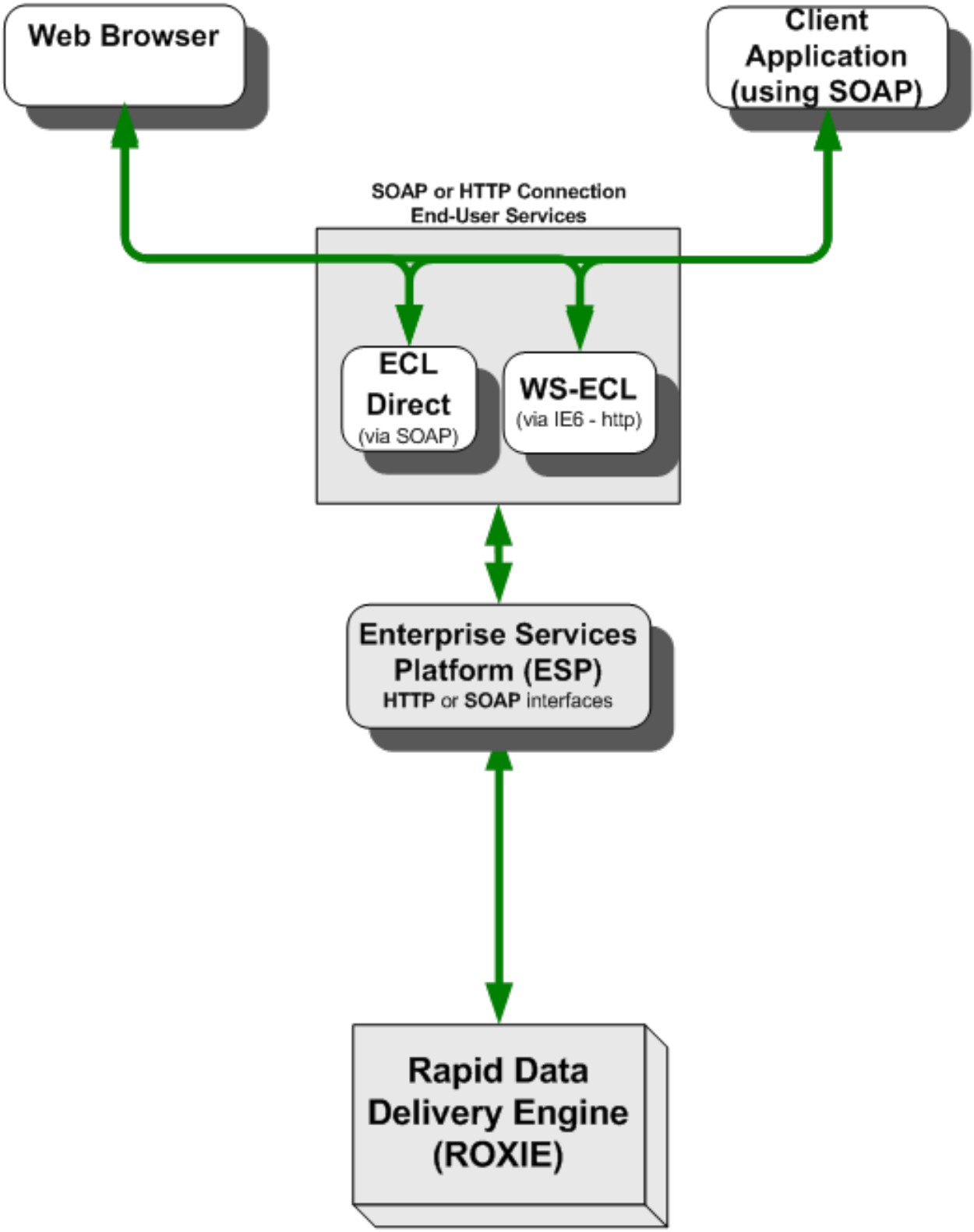
Direct Socket Connection (TCP/IP)



SOAPCALL via ECL



SOAP or HTTP/HTTPS



Managing Queries

Roxie queries are managed via Query Sets which are stored in Dali. Query Sets control which queries are loaded onto Roxie when it starts up and are manipulated by adding or removing them as required. The list of queries currently held within a QuerySet can be viewed using ECL Watch.

Once a query has been published to the Query Set, it can now be run on the roxie using a web interface.

Adding a roxie query to the Query Set

To add a roxie query to a Query Set:

1. Use ECL IDE to write your query and then compile it setting the target as the roxie cluster in your HPCC environment.

2. Go the ECL Watch tab for the compiled workunit and click the Publish button. A message is displayed indicating that your query has been published successfully.

Viewing Query Sets using ECL Watch

Using ECL Watch, you can view the query for all clusters on your HPCC. Click on the **Query Sets/Browse** menu item to see the clusters which are currently using Query Sets. Click on the **myroxie** link to view the list of queries currently available for roxie. Using this feature you can:

- View the list of currently available queries on a cluster.
- View details about each query including the ID, the name of the query, the workunit ID, the DLL(s) it uses and whether it is suspended.
- View details of the aliases that exist for each query.
- Delete a Query from the list.
- Delete an Alias.
- Toggle the suspend setting on/off

Using WsECL to run a roxie query

WsECL is the ECL Web Service interface that is provided with HPCC and is available using the following URL: http://nnn.nnn.nnn.nnn:8002, where nnn.nnn.nnn.nnn is the IP address of your ESP.

WsECL uses the Query Sets information to display the list of available runnable queries and you can use it, for example, to test that your query works as expected.

The web page shows all clusters using Query Sets. Expand myroxie in the tree and select the query you want to run. A default form is displayed which is generated from the input field names and types. Enter values and press submit to see the results and test your query.

Packages and Packagemaps

A Roxie query has one or more data files associated with it. At times, you may want to update data without changing the query.

Packages are a way to separate the data definition from the query.

A package defines a superkey that a query will use. A newer package that redefines the contents of that superkey can later be sent to a Roxie cluster without recompiling any queries. This allows you to refresh data and while ensuring that you are using identical code in your query. It also eliminates the need to recompile it and republish it.

This simplifies change control and allows query developers to continue development without the risk of a query being deployed to production before it is ready while still allowing data to be updated.

A packagemap provides a reference to the contents of a superkey used in queries that overrides the original definition . This affords greater flexibility and control over the data used by a query or a collection of queries.

Roxie resolves data files at query load time. First, it will look for a package defining the superkey contents. If there is no package, it will look in the Dali Server's DFU (Distributed File Utility).

The end result is quicker, more flexible process, without the need of recompiling complex queries. This allows you to update data without the chance of deploying new code that has not been approved for migration to production.

Without packages, you would do one of the following:

- Add new subfile(s) to the superkey via ECL code, ECLWatch, or DFUPlus, then reload the cluster.

- Revise the query to use a different key file, compile it, and republish it.

A packagemap file can contain one or more packages. The file is used to transfer the information to the Dali server. Once it is sent, the file is no longer used. It is a good idea to keep a copy as a backup, but it serves no other purpose.

The definition of a superfile or superkey inside of a package file overrides the definition in Dali, but does NOT change the superfile or superkey definition in Dali Server's DFU.

Package information is used as soon as it is added to Dali and the package is activated. This can be done using using **ecl packagemap add --activate** (or **ecl packagemap add** & **ecl packagemap activate**).

Example

In this example, we have a query named MyQuery which uses a superkey named MyDataKey that includes two subfiles:

- **~thor::Mysubfile1**
- **~thor::Mysubfile2**

If we wanted to add a third subfile, we can use a packagemap to redefine the MyDataKey superkey definition so it contains three files:

- **~thor::Mysubfile1**
- **~thor::Mysubfile2**
- **~thor::Mysubfile3**

Example 1:

```
<RoxiePackages>
 <Package id="MyQuery">
  <Base id="thor::MyData_Key"/>
 </Package>
 <Package id="thor::MyData_Key">
  <SuperFile id="~thor::MyData_Key">
   <SubFile value="~thor::Mysubfile1"/>
   <SubFile value="~thor::Mysubfile2"/>
  </SuperFile>
 </Package>
</RoxiePackages>
```

Example 2:

```
<RoxiePackages>
 <Package id="MyQuery">
  <Base id="thor::MyData_Key"/>
 </Package>
 <Package id="thor::MyData_Key">
  <SuperFile id="~thor::MyData_Key">
   <SubFile value="~thor::Mysubfile1"/>
   <SubFile value="~thor::Mysubfile2"/>
   <SubFile value="~thor::Mysubfile3"/>
  </SuperFile>
 </Package>
</RoxiePackages>
```

Updating Data

This section details the typical steps a query developer and a data developer follow once a query is ready for production.

- Prepare the data (in this workflow, the data is defined as a superkey)
- Write the query and test
- Publish a query using the data

Later when you want to update the data:

Deploy the data in one of the following manners:

- Create a packagemap containing a package redefining the contents of the superkey
- Add the packagemap(s) by associating the package information with a QuerySet.
- Add the packagemap(s) by associating the package information with a QuerySet.
- Use the command

```
ecl packagemap add --activate
```

Later when new data arrives, follow these steps to update the data using packages:

- Prepare the data and create a new subfile

Note: We strongly recommend against reusing a file name. It is generally better to create new files. Roxie shares file handles so trying to change an existing file while it is loaded could cause issues.

- Create a package with a superkey definition that includes the new subfile
- Add the packagemap redefining the contents of the superkey. Use the command:

```
ecl packagemap add --activate
```

Package File Syntax

Package files are formatted in XML using the conventions detailed in this section.

A Package file must begin with **<RoxiePackages>** and end with **</RoxiePackages>**.

Package tags have an **id** attribute that specifies what the package is referring to.

Inside the **<Package>** structure, references are made either to superfiles or other named packages. This indirect naming convention allows you to group file definitions together and reference the package containing the group.

The example shows query references first and file references following; however, this order is not required.

The lines are numbered only for reference purposes. The comments are included for clarity, but are not required.

Example:

```
 1.  <RoxiePackages>
 2.   <!-- Begin Queries -->
 3.     <Package id="MyQuery">
 4.     <Base id="thor::MyData_Key"/>
 5.    </Package>
 6.    <!-- End Queries -->
 7.    <!-- Begin File references -->
 8.    <Package id="thor::MyData_Key">
 9.      <SuperFile id="~thor::MyData_Key">
10.       <SubFile value="~thor::Mysubfile1"/>
11.       <SubFile value="~thor::Mysubfile2"/>
12.        <SubFile value="~thor::Mysubfile3"/>
13.      </SuperFile>
14.    </Package>
15.    <!--End File references -->
16.  </RoxiePackages>
```

In this example, we have a query: **MyQuery**. The query uses a superkey defined in a package named **thor::MyData_Key**. This is later defined on line 8. The **thor::MyData_Key** package contains one superkey definition.

The example shows query references first and file references following; however, this order is not required.

Working with Packages using the ecl command line tool

This section contains details about the actions and options used to work with packages. The ECL command line tool is fully documented in the Client Tools Manual.

ecl packagemap add

ecl packagemap add [--daliip][options] <target> <filename>

Examples:

```
ecl packagemap add  -s=192.168.1.10 roxie mypackagemap.pkg
ecl packagemap add roxie mypackagemap.pkg --overwrite
ecl packagemap add roxie mypackagemap.pkg --daliip=192.168.11.11
```

ecl packagemap add	Calls the packagemap add command
Actions	
add	Adds a packagemap to the target cluster
Arguments	
target	The target to associate the packagemap with
filename	The name of the file containing packagemap information.
--daliip=	IP address or hostname of the remote Dali to use for logical file lookups
Options	
-O, --overwrite	Whether to overwrite existing information - true if present
-A, --activate	Activates packagemap
-v, --verbose	Output additional tracing information
-s, --server	The IP Address or hostname of ESP server running eclwatch services
--port	The eclwatch services port (Default is 8010)
-u, --username	The username (if necessary)
-pw, --password	The password (if necessary)

ecl packagemap delete

ecl packagemap delete [options] <target><packagemap>

Examples:

```
ecl packagemap delete roxie mypackagemap
```

ecl packagemap delete	Calls the packagemap delete command
Actions	
delete	Deletes a packagemap
Options	
-v, --verbose	Output additional tracing information
-s, --server	The IP Address or hostname of ESP server running eclwatch services
--port	The eclwatch services port (Default is 8010)
-u, --username	The username (if necessary)
-pw, --password	The password (if necessary)

ecl packagemap activate

ecl packagemap activate <target> <packagemap>

Example:

```
ecl packagemap activate roxie mypackagemap.pkg
```

ecl packagemap activate	The activate command will deactivate the currently active packagemap and make the spec packagemap active.
Arguments	
target	The target containing the packagemap to activate
packagemap	name of packagemap to update
Options	
-v, --verbose	Output additional tracing information
-s, --server	The IP Address or hostname of ESP server running eclwatch services
--port	The eclwatch services port (Default is 8010)
-u, --username	The username (if necessary)
-pw, --password	The password (if necessary)

ecl packagemap deactivate

ecl packagemap deactivate <target> <packagemap>

Example:

```
ecl packagemap deactivate roxie mypackagemap.pkg
```

ecl packagemap deactivate	The deactivate command will deactivate the currently active packagemap.
Arguments	
target	The target containing the packagemap to deactivate
packagemap	Name of packagemap to deactivate
Options	
-v, --verbose	Output additional tracing information
-s, --server	The IP Address or hostname of ESP server running eclwatch services
--port	The eclwatch services port (Default is 8010)
-u, --username	The username (if necessary)
-pw, --password	The password (if necessary)

ecl packagemap list

ecl packagemap list <target>

Examples:

```
ecl packagemap list roxie
```

ecl packagemap list	Calls the packagemap list command
Actions	
list	Lists loaded packagemap names
Arguments	
target	The target containing the packagemap to list
Options	
-v, --verbose	Output additional tracing information
-s, --server	The IP Address or hostname of ESP server running eclwatch services
--port	The eclwatch services port (Default is 8010)
-u, --username	The username (if necessary)
-pw, --password	The password (if necessary)

ecl packagemap info

ecl packagemap info [options] <target>

Examples:

```
ecl packagemap info roxie
```

ecl packagemap info	Calls the packagemap info command
Actions	
info	returns packagemap info
Arguments	
target	The target containing the packagemap to retrieve
Options	
-v, --verbose	Output additional tracing information
-s, --server	The IP Address or hostname of ESP server running eclwatch services
--port	The eclwatch services port (Default is 8010)
-u, --username	The username (if necessary)
-pw, --password	The password (if necessary)

ecl packagemap validate

ecl packagemap validate <target> [<filename>]

Examples:

```
ecl packagemap validate roxie mypackagemap.pkg
ecl packagemap validate roxie --active
```

The packagemap validate command verifies that :

- Referenced superkeys have subfiles defined (warns if no subfiles exist)
- All referenced queries exist in the current Roxie queryset
- All Roxie queries are defined in the package

The result will also list any files that are used by queries but not mapped in the packagemap.

Filename, --active, and --pmid are mutually exclusive. The --active or --pmid options validate a packagemap that has already been added instead of a local file.

The --queryid option checks the files in a query instead of all the queries in the target queryset. This is quicker when you only need to validate the files for a single query.

ecl packagemap validate	Calls the packagemap validate command.
Actions	
validate	Validates packagemap info
Arguments	
filename	The filename containing the packagemap info to validate
target	The target containing the packagemap to validate
Options	
-v, --verbose	Output additional tracing information
-s, --server	The IP Address or hostname of ESP server running eclwatch services
--active	Validates the packagemap that is active for the given target
--pmid=<packagemapid>	Validates the given packagemap
--queryid	Validate the files for the given queryid if they are mapped in the packagemap
--port	The eclwatch services port (Default is 8010)
-u, --username	The username (if necessary)
-pw, --password	The password (if necessary)

Tips:

- Always use superfiles or superkeys to take advantage of indirect naming and to allow the use of packages. Roxie does not require this, but it works best this way.
- Use unique filenames instead of overwriting existing files. This prevents accidental overwrites and provides an easy way to roll back.
- If you have a large collection of keys, it is easier to maintain if you use superkeys with a single subindex-file. Multiple subfiles are useful when you need to add data quickly, but if time allows, it is better to rebuild a new single key file.
- Before you delete a packagemap, make sure you have a backup copy.

Deploying Data to a Cluster using DFU

You can use the Distributed File Utility (DFU) in ECL Watch to copy (or remote copy) data files to a Roxie cluster. This allows you to copy large files to a cluster in advance of publishing a query. If data files are copied in advance, a query which requires those files will use the ones already in place. If you have large data sets, this allows you to prepare in advance of query deployment.

Note: To use this feature, the FTSlave utility must be installed to each node in the cluster. This is done automatically by the Configuration Manager Wizard.

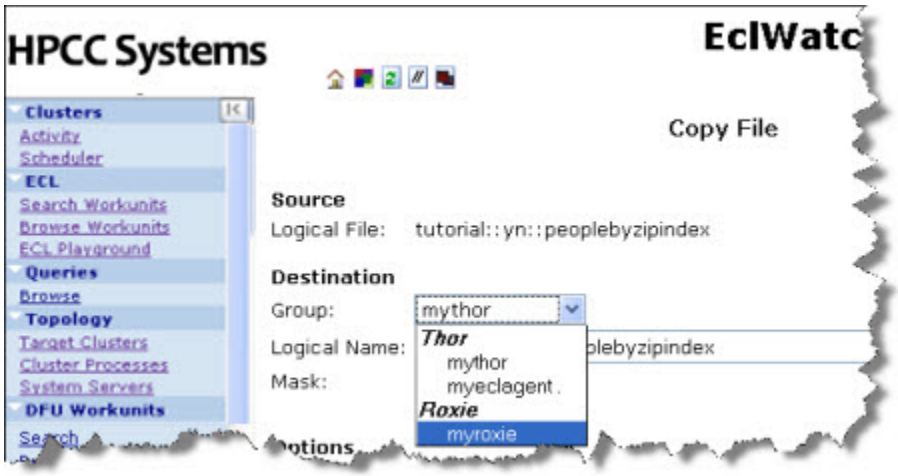
DFU Copy

1. Open the ECLWatch web page. You can reach this page using the following URL: http://nnn.nnn.nnn.nnn:8010, where nnn.nnn.nnn.nnn is your node's IP address.

2. Click on **Browse Logical Files** hyperlink below **DFU Files** in the menu on the left.

3. Locate the file to copy in the list of files, then click on the icon, then select **Copy** from the pop-up menu.

4. Fill in **Destination** and **Options**information.



	Destination:	
Group		Use the drop list to select the Roxie cluster to copy to.
	Note:	You can only choose from clusters within the current environment.
Logical File		The logical name for the copied file.
File Mask		Automatically updated based on logical file name entered.
	Options:	
Overwrite		Check this box to overwrite files of the same name.
Replicate		Check this box to create backup copies of all file parts in the backup directory (by convention on the secondary drive of the node following in the cluster). **This option is only available on systems where replication has been enabled.**

Wrap	Check this box to keep the number of parts the same and wrap if the target cluster is smaller that the original.
Compress	Check this box to compress the files.

5. Press the **Submit** button.

   The **DFU Workunit**displays.

6. Click on the **View Progress** hyperlink to see a progress indicator.

Remote Copy

Remote Copy allows you to copy data to a Roxie from a Thor or Roxie cluster outside your environment.

1. Open the ECLWatch web page. You can reach this page using the following URL: http://nnn.nnn.nnn.nnn:8010, where nnn.nnn.nnn.nnn is your node's IP address.

2. Click on the **Remote Copy** hyperlink below **DFU Files** in the menu on the left.

   The **Copy File** page displays.

3. Fill in **Source, Destination,** and **Options**information.

Source:	
Logical File	The logical file name in the remote environment
Source Dali	The Dali Server in the remote environment
Source Username	A valid user in the remote environment
Source Password	The password for the user in the remote environment
Destination:	
Group	Use the drop list to select the cluster to copy to.
Note:	You can only choose from clusters within the current environment.
Logical File	The logical name for the copied file.
File Mask	Automatically updated based on logical file name entered.
Options:	
Overwrite	Check this box to overwrite files of the same name.
Replicate	Check this box to create backup copies of all file parts in the backup directory (by convention on the secondary drive of the node following in the cluster). **This option is only available on systems where replication has been enabled.**
Compress	Check this box to compress the files.
Wrap	Check this box to keep the number of parts the same and wrap if the target cluster is smaller that the original.

4. Press the **Submit** button.

   The **DFU Workunit** displays.

5. Click on the **View Progress** hyperlink to see a progress indicator.

Capacity Planning for Roxie Clusters

Capacity Planning

Roxie clusters are disk-based High Performance Computing Clusters (HPCC) , typically using indexed files. A cluster is capable of storing and manipulating as much data as its combined hard drive space; however, this does not produce optimal performance.

For maximum performance, you should configure your cluster so slave nodes perform most jobs in memory.

For example, if a query uses three data files with a combined file size of 60 GB, a 40-channel cluster is a good size, while a 60-channel is probably better.

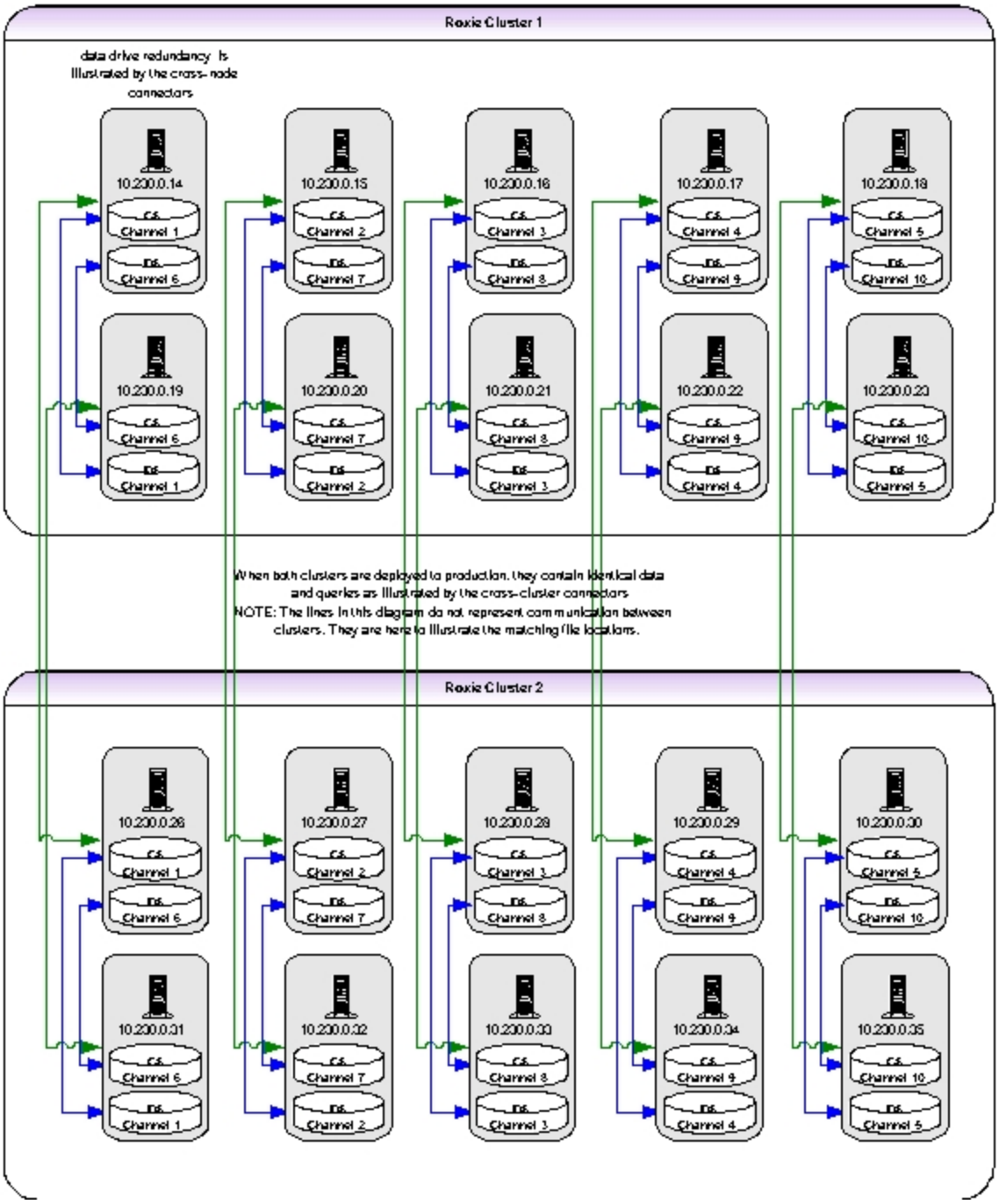
Another consideration is the size of the Thor cluster creating the data files and index files to be loaded. Your target Roxie cluster should be the same size as the Thor on which the data and index files are created or a number evenly divisible by the size of your Roxie cluster. For example, a 100-way Thor to a 20-way Roxie would be acceptable.

This is due to the manner in which data is loaded and processed by Roxie slaves. If data is copied to slave nodes, the file parts are directly copied from source location to target locations. They are NOT split or resized to fit a different sized cluster. Therefore, if you load 50 file parts onto a 40-channel cluster, part one goes to channel one, part two to channel two, etc. Parts 41-50 start at the top again so that part 41 goes to channel 1, and part 42 goes to channel 2, etc. The result is an unevenly distributed workload and would result in reduced performance. A cluster will only perform as fast as its slowest node.

The final consideration is the number of Server processes in a cluster. Each slave must also be a Server, but you can dedicate additional nodes to be only Server processes. This is useful for queries that require processing on the Server after results are returned from slaves. Those Server-intensive queries could be sent only to dedicated Server IP addresses so the load is removed from nodes acting as both Server and slave.

Configuring the Channels

In the illustration below, the nodes are configured using an N+5 scheme to share channels. Channels can be configured in many ways, this is one example.



In this depiction, each enclosure holds five Roxie slave blades (a row of servers in the picture). We will use this example for the rest of this manual.

PreFlight and Roxie Metrics

ECL Watch / Preflight

ECL Watch's Topology section is used to perform Preflight activities. These preflight utilities are used for daily health checks, as well as trouble avoidance and troubleshooting. It provides a central location to gather hardware and software information from a remote set of machines and has many uses for day-to-day environment preparation.

Regular Uses

Typical uses for Preflight in the preparation of the environment before a data build

- Examine disk space utilization
  - Critical for optimal operation of the Data Refinery nodes.
  - Disk space utilization alarm threshold easily adjusted.
- View last boot up of all nodes in a cluster
  - Confirms the boot-up state of a node without having to login to the node.
- Confirm expected processes running on each node
  - Make sure all HPCC processes are running on the appropriate nodes
- Validate network availability of each node

Using ECL Watch PreFlight in Troubleshooting

These are typical uses for Preflight in troubleshooting

- Validate Disk Space Utilization
- Validate Memory Utilization
- Validate monitoring agent process running on nodes (heimdall.exe)
- Validate processes running on the appropriate machines

On a routine basis, typically before processing begins, an entire system health check should be performed.

There are three sections in the Topology area of ECL Watch:

- Clusters, including all clusters of any of the following types:
  - Thor (Data Refinery Cluster)
  - Roxie (Rapid Data Delivery Engine Cluster)
- System Servers, including all system components in the current environment:
  - Dali Server
  - DFU Server(s)

- ECL CC Server(s)
- ECL Scheduler
- ECL Agent(s)
- ESP Server(s)
- Sasha Server

ECL Watch PreFlight Tasks

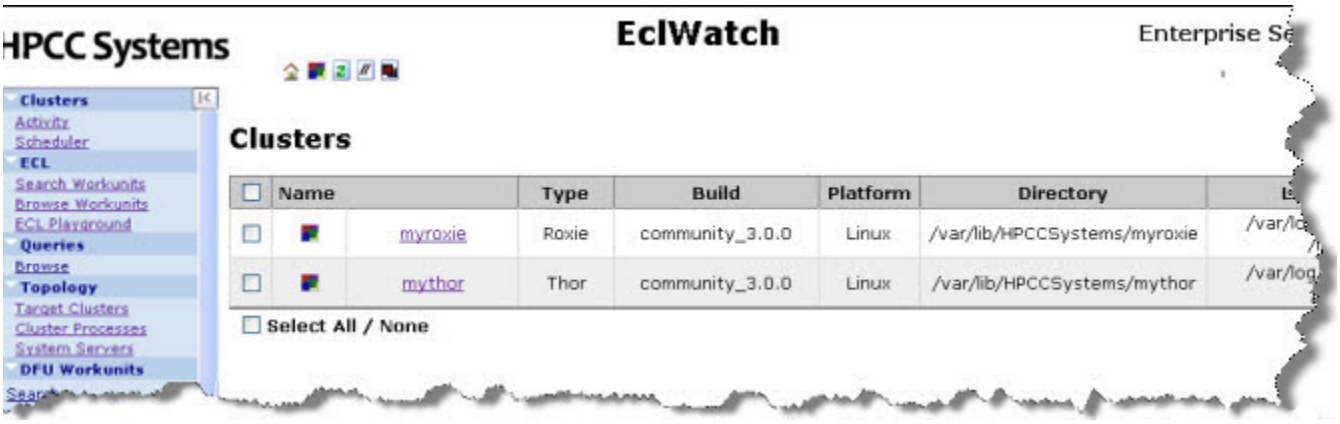
Perform these tasks using ECL Watch when validating or troubleshooting a component.

- Open the ECLWatch web page. You can reach this page using the following URL: http://nnn.nnn.nnn.nnn:8010, where nnn.nnn.nnn.nnn is your node's IP address.

- In the left side menu, click on one of the hyperlinks under **Topology**:

  System Servers or **Clusters**

- For a cluster, click on its hyperlink.



A page displays showing all machines of the type selected in the previous steps.

- For a cluster, make sure the **Select All** checkbox is checked (to select all servers).

  For System Servers, the critical servers are pre-selected. If you only want to PreFlight a specific machine, check only the box next to its listing.

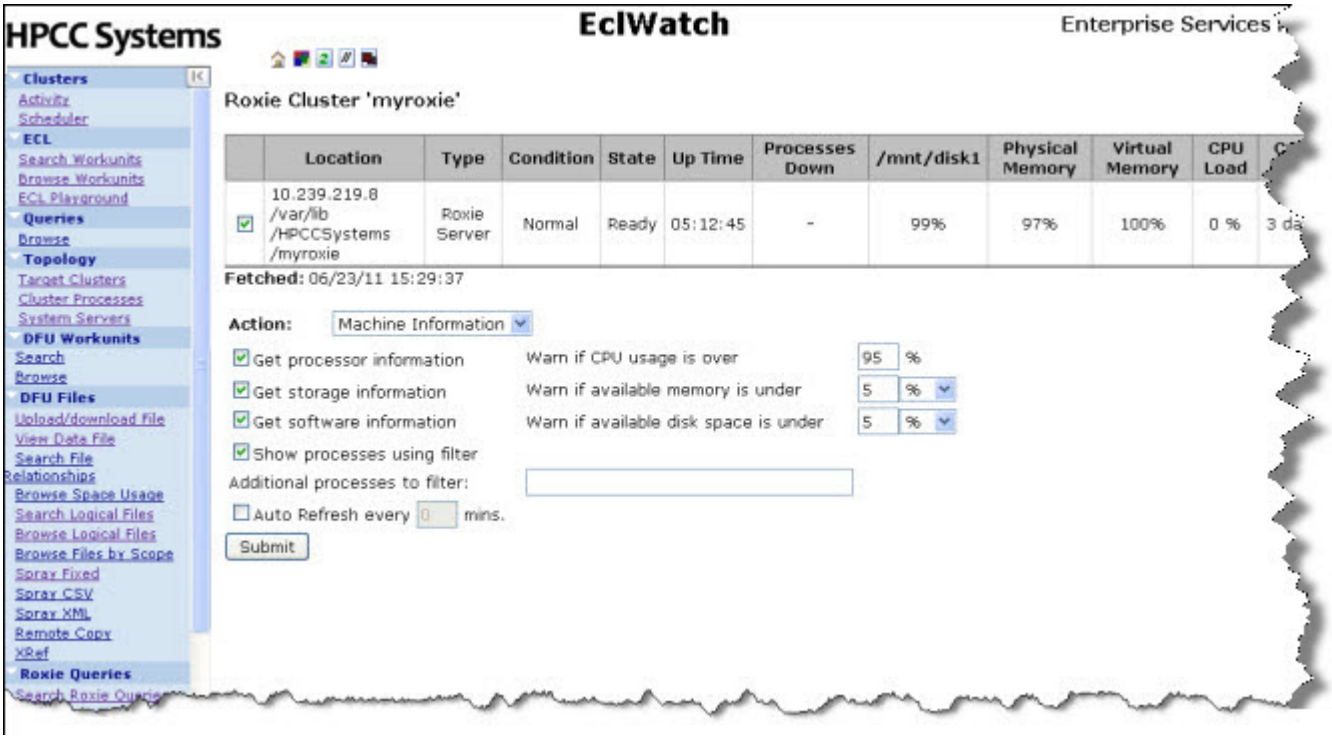
- Check the boxes for **Get Processor Information**, **Get storage information**, and **Get software information**. These should be checked automatically.

- Check the **Show processes using** filter checkbox. Submitting a request using the predefined filters verifies whether the appropriate HPCC processes are running. For example, a machine defined as a Dali server would be verified for all processes expected to run on a Dali server.

- Optionally, check the **Auto Refresh Every** checkbox and provide the number of minutes. This will automatically query the servers and refresh the page at the provided interval.

- Press the **Submit** button.

  The resulting page displays all the requested information, as shown below:



- Any expected processes that are not running are highlighted in orange in the **Processes Down** column.
- Hovering the mouse over any column, displays additional information.
- Any values below the specified thresholds (see above) are displayed with an orange background.
- For disk space utilization, all local drives are displayed. Partitions to exclude (such as RAM Drives) are defined in the ECLWatch service.

Options

The following option settings are available to customize the execution and display.

Machine Information

Specifies that the query will return machine information for the selected machines.

Get Roxie Metrics

This option is available only to Roxie cluster. It displays diagnostic information about the cluster. See the *Roxie Metrics*section in this manual.

Get Processor Information

Specifies that the query will include results about the processor usage on the selected machines. The threshold for a warning is set in the **Warn if CPU Usage is over** entry control.

Get storage Information

Specifies that the query will include results about memory and disk space on the selected machines. The threshold for a warning about memory is set in the **Warn if available memory is under** entry control. The threshold for disk space warnings is set in the **Warn if available disk space is under** entry control.

Get software Information

Specifies that the query will include results about running processes on the selected machines.

Show Processes using filter

Checking this box specifies that a query using predefined filters will verify whether the appropriate HPCC processes are running. For example, a machine defined as a Dali server would be verified for all processes expected to run on a Dali server. If not checked, all running processes are displayed in a pop-up window when the mouse pointer is over the column.

Additional Processes to filter

When using a filter, this entry box allows you to add additional processes to consider. If the specified process is not running, it will appear in the **Down Processes** column.

Auto Refresh Every ___ mins.

Optionally, provide an interval in minutes to automatically re-submit the query.

Roxie Queries and Roxie Files in ECL Watch

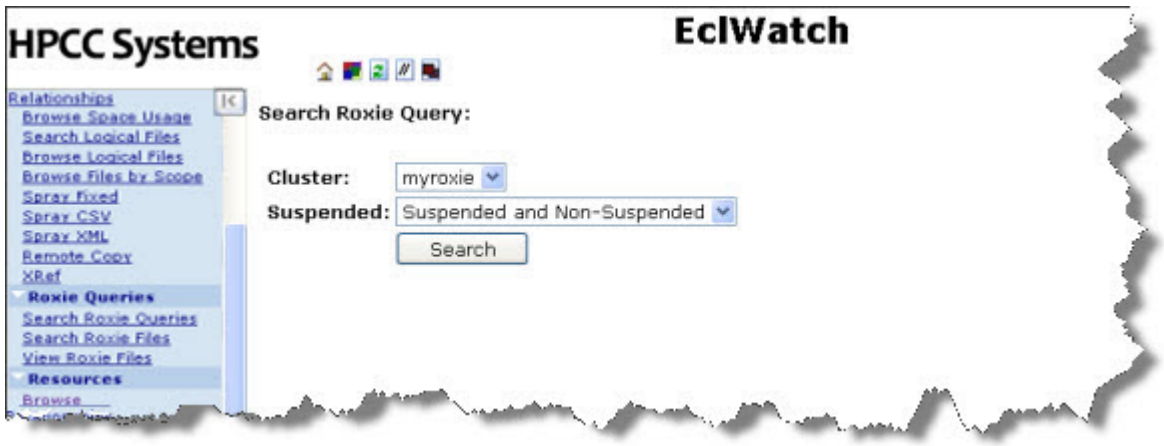
ECL Watch allows you to search for deployed Roxie Queries and to search for deployed Roxie Files.

To Search Roxie Queries

- Open the ECLWatch web page. You can reach this page using the following URL: http://nnn.nnn.nnn.nnn:8010, where nnn.nnn.nnn.nnn is your node's IP address.

The ECL Watch home page will open.

- Click on the **Search Roxie Queries** below **Roxie Queries**



- Select the **Cluster** from the drop list.

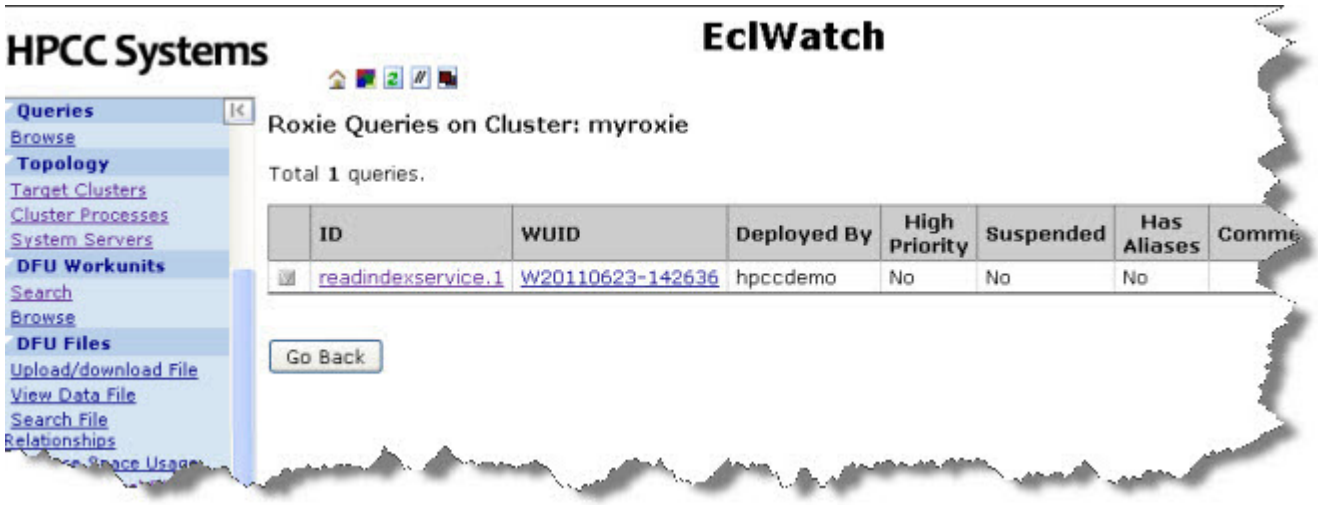
All Roxie clusters in this environment appear on the list.

- Select the query type (Suspended, Non-Suspended, or both) from the **Suspended** drop list.

- Press the **Search** button

The list of queries displays.

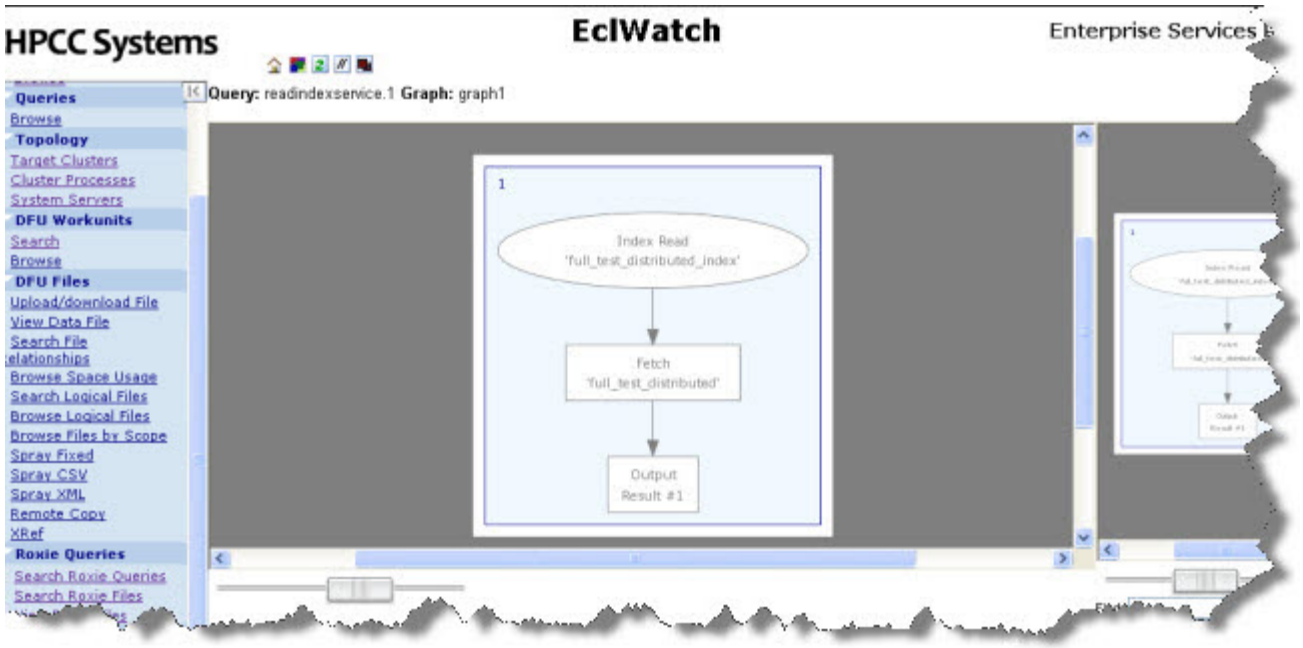
- Rt-click to the button on the left of any query, then select the action from the popup menu.



Details displays a page of details about the query, including:

- The Workunit ID
- The Cluster
- The name of the associated Shared Object (.so)
- The Query's priority
- The user that deployed it
- Whether or not it is suspended.
- **ShowSuperFiles** lists all superfiles used by the query
- **ShowNonSuperFiles** lists all non-superfiles used by the query
- **ShowGraph** displays the execution graph for the query.

- **ShowGVCGraph** displays the execution query's graph (in GVC format).



To Search Roxie Files:

- Open the ECL Watch home page in a browser. The URL is the IP address of your ESP Server plus the port to which the EclWatch Service is bound (usually 8010).

The ECL Watch home page will open.

- Click on the **Search Roxie Files** below **Roxie Queries** in the menu on the left side.

- Select the **Cluster** from the drop list.

All Roxie clusters in this environment appear on the list.

- Select the file type (Non-Super Files or Super FIles) from the **File Type** drop list.
- Press the **Search** button

The list of files displays.

- Rt-click to the button on the left of any file, then select the action from the popup menu.

- **Details** displays a page of details about the file.
- **ViewDataFilee** allows you to view the contents of a file.
- **ShowQueries** lists all queries using this file

Viewing Roxie Key Files

- Open the ECL Watch home page in a browser. The URL is the IP address of your ESP Server plus the port to which the EclWatch Service is bound (usually 8010).

The ECL Watch home page will open.

- Click on the **Search Roxie Files** below **Roxie Queries** in the menu on the left side.

- Select the **Cluster** from the drop list.

All Roxie clusters in this environment appear on the list.

- Select the file type (Non-Super Files or Super FIles) from the **File Type** drop list.
- Press the **Search** button.

The list of files displays.

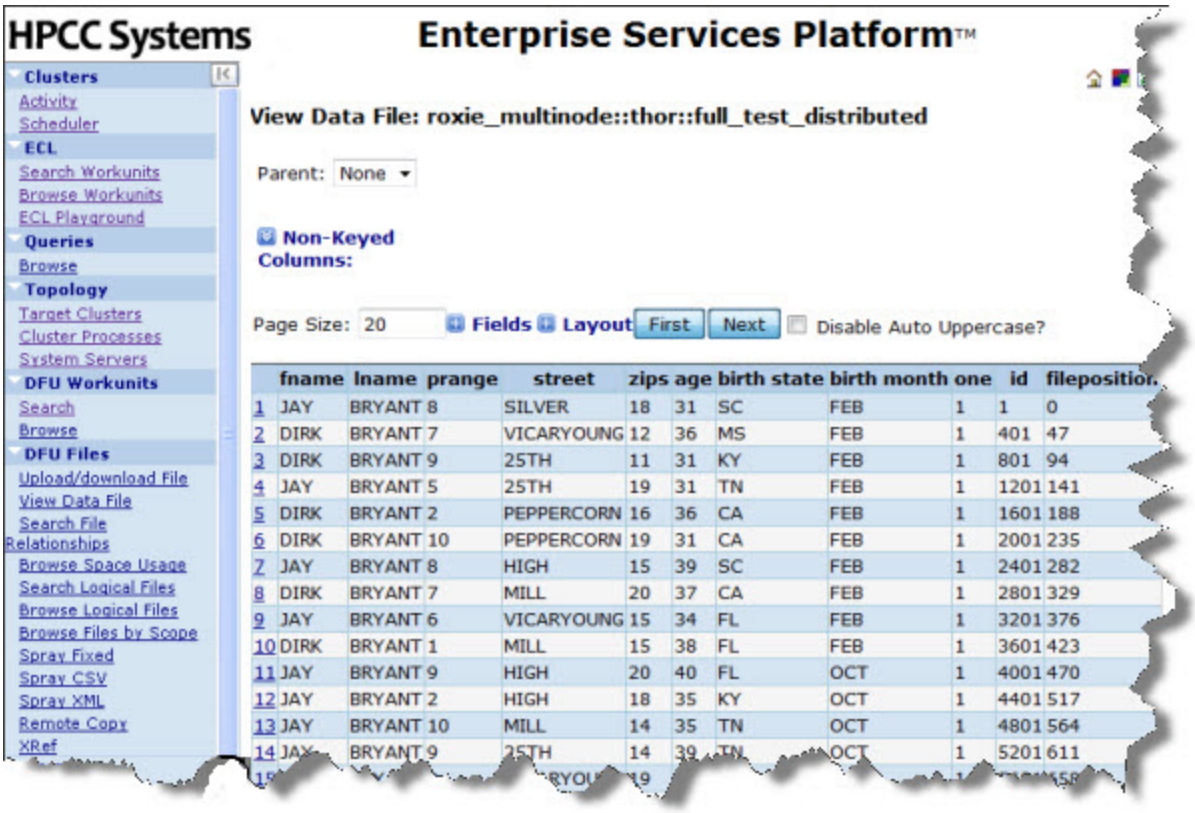
- Rt-click to the  button on the left of any key file, then select **ViewDataFile** from the popup menu.

A form displays (containing fields in the Key File)

- Provide search criteria in the fields, then press the **First** button.

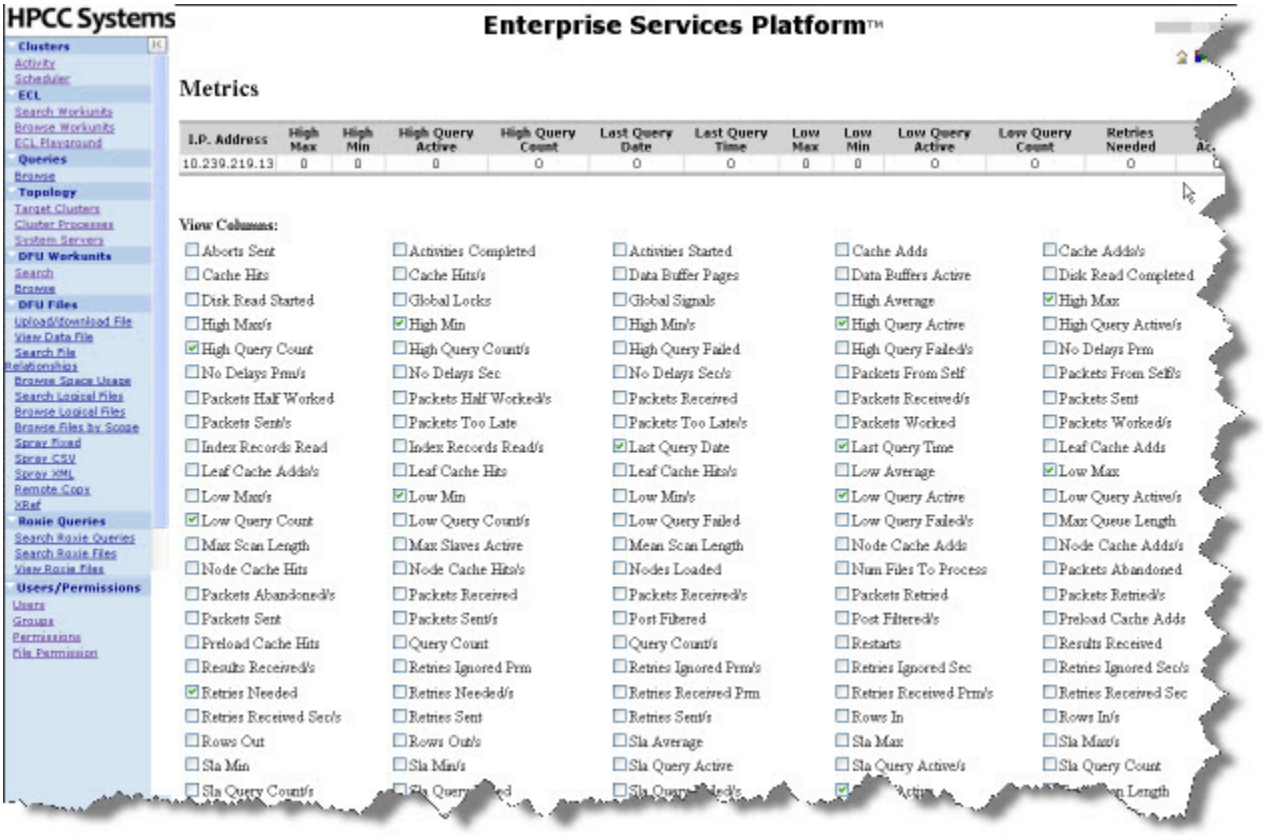
Searching using a value in a Keyed column performs an indexed read. Values in Non-keyed columns provide post-fetch filtering (after the indexed read).

The result displays in a similar manner as shown below:



	fname	lname	prange	street	zips	age	birth state	birth month	one	id	fileposition
1	JAY	BRYANT	8	SILVER	18	31	SC	FEB	1	1	0
2	DIRK	BRYANT	7	VICARYOUNG	12	36	MS	FEB	1	401	47
3	DIRK	BRYANT	9	25TH	11	31	KY	FEB	1	801	94
4	JAY	BRYANT	5	25TH	19	31	TN	FEB	1	1201	141
5	DIRK	BRYANT	2	PEPPERCORN	16	36	CA	FEB	1	1601	188
6	DIRK	BRYANT	10	PEPPERCORN	19	31	CA	FEB	1	2001	235
7	JAY	BRYANT	8	HIGH	15	39	SC	FEB	1	2401	282
8	DIRK	BRYANT	7	MILL	20	37	CA	FEB	1	2801	329
9	JAY	BRYANT	6	VICARYOUNG	15	34	FL	FEB	1	3201	376
10	DIRK	BRYANT	1	MILL	15	38	FL	FEB	1	3601	423
11	JAY	BRYANT	9	HIGH	20	40	FL	OCT	1	4001	470
12	JAY	BRYANT	2	HIGH	18	35	KY	OCT	1	4401	517
13	JAY	BRYANT	10	MILL	14	35	TN	OCT	1	4801	564
14	JAY	BRYANT	9	25TH	14	39	TN	OCT	1	5201	611

Roxie Metrics



IP Address	The IP address of the Server node
slavesActive	Number of slaves active for that Server
lastQueryTime	Time stamp of most recent query executed
loQueryActive	Number of low priority queries active
loMin	Minimum time (ms) it took to run a low priority query
retriesNeeded	Count of all reply packets that arrived to a Server as a response to more than one try.
loMax	Maximum time (ms) it took to run a low priority query
hiQueryActive	Number of high priority queries active
loQueryCount	Total number of low priority queries run
loQueryAverage	Average time it took to run a high priority query
hiQueryCount	Total number of high priority queries run
hiMax	Maximum time (ms) it took to run a high priority query
hiMin	Minimum time (ms) it took to run a high priority query
heapBlocksAllocated	Number of times Roxie had to allocate memory from its memory allocator