HPCC SYSTEMS

LexisNexis® RISK SOLUTIONS

WHITE PAPER

Understanding HPCC Systems and Spark - A Comparative Analysis

OVERVIEW

Technologies like social media, e-commerce, and the Internet of Things (IoT) are generating massive amounts of data. According to a 2022 report from Statista, global data creation is projected to grow to over 180 zettabytes by 2025. This kind of dramatic growth creates problems for organizations as they work to capture, classify, and analyze the data generated by customers and their own employees and operations.

To address the data challenge, many organizations are implementing data lake platforms: a collection of software tools for ingesting, processing, delivering, and governing complex data, plus the hardware used to process and store the data (stored on-premises in a datacenter or off-premises in a public or private cloud). Choosing the right platform can be a daunting process; there are many options available to organizations looking to implement data lakes, and each has its strengths and weaknesses.

Apache Spark is a widely used, market-leading data lake processing engine with a robust developer ecosystem and a large install base. That said, Spark requires a lot of optimization and customization, and as Spark was not originally developed for end-to-end data lake management, much of its functionality comes from software developed by other companies. The complexity, costs, and time associated with implementing Spark as the processing engine for an end-to-end data lake platform can be challenging for smaller or less experienced IT teams. Thankfully, there is an alternative to Spark that can provide a complete data lake platform that is both less costly and requires much less time to implement. That platform is HPCC Systems.

The beginnings of the technological foundation for the HPCC Systems platform were first developed at Seisint in 1999 as a memory-based system to query large amounts of data. In 2000, the team at Seisint needed to collect and analyze massive amounts of raw consumer financial data from a wide variety of sources for a customer responsible for determining consumer credit scores in the United States. This led to the creation of the programming language, known as ECL (Enterprise Control Language), that HPCC Systems uses today. In 2004, LexisNexis® Risk Solutions (LNRS) acquired Seisint and its foundational technology. Initially, HPCC Systems mainly served as an internal tool for LexisNexis Risk Solutions

and had not yet reached its full potential. In 2008, LexisNexis Risk Solutions acquired ChoicePoint, an insurance analytics provider, and over the next three years, ChoicePoint's product portfolio was integrated into the HPCC Systems platform. Combining the HPCC Systems platform with the ChoicePoint insurance products created powerful business advantages -- an extremely efficient big data processing solution capable of managing massive amounts of data and driving much more efficient data products into the already massive insurance market. In 2011, LexisNexis Risk Solutions decided to release HPCC Systems under an open source license. Since its launch, HPCC Systems has developed a rich user community and a global network of customers that currently support a multibillion-dollar business. Additionally, HPCC Systems continues to innovate the platform by adding support for cloud-based environments; developing new data cataloging, governance, and orchestration tools; and adding new administration and dashboard reporting capabilities. Current HPCC Systems users include Quod (the new credit bureau for Brazil that LNRS helped to create), and many prominent universities: Clemson University, Florida Atlantic University, Kennesaw State University, RV College of Engineering, and several other universities globally.

Since its beginning, HPCC Systems has given its users a platform consisting of a single homogenous data pipeline. This significantly minimizes the amount of effort users spend on platform management, installation, and maintenance. Perfect for both data lakes and warehouses, HPCC Systems is extremely capable and efficient in processing large amounts of data due to an architectural design that leverages two specialized clusters, named Thor and Roxie, to manage and optimize the platform's various functions.

This paper serves as a comparison between the architectures and feature support of Spark and HPCC Systems in regard to data lake capabilities. It is not meant to provide a performance comparison between the two platforms. To see a comprehensive performance comparison of the two, including benchmarks for specific tasks in the data pipeline, please refer to the HPCC Systems technical white paper, Comparison of HPCC Systems Thor vs Apache Spark Performance with AWS.

General Conceptual Differences

Spark and HPCC Systems were developed using different design philosophies. Spark was designed to be a part of a larger "big data" ecosystem and doesn't offer solutions or systems for parts of a large-scale data lake platform. For example, Spark has no built-in support for user management, data storage, data delivery, execution control, or administration tools. HPCC Systems was designed to address user needs around data delivery, and places significant importance on enabling a complete data lake platform in a single package. This is illustrated by the short amount of time it takes to bring an HPCC Systems big data solution online. HPCC Systems solutions can be up and running in a matter of hours, while a Spark-centric solution can take weeks to configure (depending on the complexity of the configuration) as developers must source the components of their data lake platform not supported by Spark from other vendors, and then configure those components to work with Spark.

Spark focuses on data science, ingestion, and ETL, while HPCC Systems is more focused on ETL, data delivery, and data governance.

Also, Spark and HPCC Systems each focus on different parts of the big data pipeline. Spark focuses on data science, ingestion, and ETL, while HPCC Systems is more focused on ETL, data delivery, and data governance. That said, potential users should realize they don't have to choose between the two; users can operate a data lake in a mixed Spark/HPCC Systems environment. Because it is a complete data lake management solution, adopting HPCC Systems to launch the data lake platform first and then integrating Spark cluster(s) to leverage its data science capabilities can be a "best of both worlds" arrangement in some cases. To facilitate such an implementation, HPCC Systems developed the Spark-HPCC Connector to allow Spark cluster(s) two-way access to data stored on an HPCC Systems cluster.

After discussing the general conceptual differences between the two platforms, this paper will compare their data ingestion, processing, delivery, and governance capabilities. The paper closes with a comparative discussion of cloud native support. A detailed examination of how each platform handles different stages in the data pipeline can help users understand the similarities and differences between the two, and also identify if one platform (or a combination of both) makes the most sense for their specific use case.

Data Ingestion Support

Both platforms support a wide variety of raw file formats during ingestion, including CSV, XML, JSON, plain text, and binary files. This makes it easier to work with different data sources.

The main difference is that since Spark is essentially a processing engine, a data lake solution built around Spark needs to be completed with a third-party data store; typically, a Hadoop Distributed File System (HDFS) to hold the raw data. In other words, Spark on its own can't act as a complete data lake platform; third-party software will need to be used to provide the end-to-end functionality that Spark lacks. Fortunately, Spark does offer public facing APIs equipped with "datasource connectors," which allow Spark to read data from other storage sources (HDFS, local disk, data lakes, S3, etc.).

HPCC Systems, on the other hand, has its own distributed file clusters, which the platform calls Thor and Roxie. Raw files are maintained in standard Linux filesystems, and data storage is implemented via a modernized version of Indexed Sequential Access Method (ISAM) files. ISAM is a system where data is stored in files on disk and search indexes are built to allow direct access to the individual records within that dataset.

A spray is copying data from one location (such as a landing zone) to a cluster.

The import of raw data into Thor happens via a process called a spray. A spray is copying data from one location (such as a landing zone) to a cluster. The term spray was adopted due to the nature of the file movement; the data in the file is partitioned across all nodes of the cluster. You can easily spray files through an HPCC Systems web browser interface (called ECL Watch) via ECL code or via a command line interface. Once a raw file is sprayed (and therefore partitioned within HPCC Systems), it becomes referenced as a logical file; a single logical entity that abstracts the partitioning from the programmer's perspective. After the content of the logical files is properly cleaned and transformed via ECL code, indexes are then usually created in the Thor cluster and replicated to Roxie clusters for fast data delivery.

HPCC Systems easily allows for the periodic ingestion of new raw data thanks to its support of SuperFiles. A SuperFile is a container of logical files (referenced to as sub-files) that is treated like a single logical entity. The major advantage of using SuperFiles is the easy maintenance of its sub-files. Indeed, updating the actual data a query reads can be as simple as adding a new sub-file to an existing SuperFile. Spark doesn't offer native support of SuperFiles, though some of its built-in datasource functions support reading multiple files in a single dataset.

When it comes to streaming data, Spark supports fault tolerance via various recovery modes. This is important because most streaming data applications run 24/7 and must be resilient to failures unrelated to the application logic (e.g., system failures, JVM crashes, etc.). However, proper implementation of fault tolerance requires that a programmer already be aware of the potential causes of a fault in real-time data processing in order to write the code to overcome them. To do this, Spark Streaming needs to *checkpoint* enough information to a fault-tolerant storage system so it can recover from failures.

HPCC Systems has a plug-in for Kafka to allow users to add streaming data support to their implementation.

Data Processing Architecture

As stated earlier, Spark and HPCC Systems were built to serve different, yet complementary, purposes. A more detailed examination of these architectural differences at the data processing level can help users determine which solution is right for them.

First, let's look at the shared similarities between the Spark and HPCC Systems architectures. Both platforms support a "distributed computing" model in which datasets are divided into different parts (larger datacenters can have hundreds or thousands of such partitions), and the compute processing that powers the manipulation of data is, whenever possible local to where the parts reside. Both HPCC Systems and Spark also allow for the use of non-local data; partitions can reside in an on-premises datacenter or be stored remotely in a public or private cloud.

The differences in Spark and HPCC Systems architectures become apparent when comparing how each platform handles computations and job execution.

Although originally written in Scala, Spark is compatible with many commonly used programming languages, including Python, JavaR, and SQL. Python and R support in particular is important as many data scientists using the Spark platform can write code in those languages. The Spark ecosystem developed APIs for Python and R that run on top of Spark, allowing for an easier transition for Python users unfamiliar with Scala. Spark implements these languages at the driver level, so they are very well integrated into the Spark platform. Spark also offers some mechanisms for Java users to incorporate Java libraries into their code.

The differences in Spark and HPCC Systems architectures become apparent when comparing how each platform handles computations and job execution.

Spark uses a programming model featuring a master/worker architecture, where the master creates a driver program responsible for managing processing jobs between partitions and creating the interface for reporting the results of those jobs. The driver program then begins processing the jobs either on the machine that requested it or on a different partition in the cluster. This architecture can be problematic when you consider Spark doesn't automatically provide a job dispatching control mechanism. This means Spark programmers must ensure when writing code that they aren't instructing the driver program in a way that will block execution on the cluster by processing one job that's reliant on another that hasn't been completed. Additionally, programmers need to ensure they aren't inadvertently using operations that cause the driver to collect a dataset that's too large (for example, inappropriate application of the dataframe.collect() command), which can impact the runtime of a job.

To keep data moving in a Spark implementation, jobs are broken down into tasks that a centralized scheduler sends to be executed on nodes in the cluster. Tasks can be run on any node, but the scheduler is aware of data locality and data requirements for each task and will attempt to run a given task on a node close to the data (i.e., in the same physical machine, server cabinet, or in some circumstances the same datacenter). Spark has an API for users who want to repartition data at runtime, as changing partitions can improve performance for certain operations, and the number of partitions doesn't have to match the number of compute planes. Ultimately, it's up to the Spark programmer to make sure the driver program is optimized to perform processing as quickly and efficiently as possible, based on the specifics of their implementation and their organization's needs.

Data used in a Spark computation is manipulated in a logical structure known as a Resilient Distributed Dataset (RDD). That data is stored in server memory, but may spill over to the hard drive in the case of larger files. RDDs are immutable; any changes to a dataset during computation will result in a new RDD being created to ensure the original data remains unchanged. RDDs store a full computation history and can be re-computed from the original source data at any time. This feature is primarily used for error recovery, but can also be used to drop datasets out of memory in order to recover memory in object-oriented modeling (OOM) scenarios. Another benefit of using RDDs is fault tolerance.

Spark is capable of identifying when a cluster fails and then shifting its assigned task to another cluster automatically; RDDs allow the data to then be recomputed on a backup cluster.

Like Spark, HPCC Systems uses a master/worker architecture. But unlike Spark, HPCC Systems automatically provides a job dispatching mechanism that can support parallel execution queues, each controlled by an individual master program that coordinates the most efficient execution of tasks in the queue. Each worker processes one portion of an input data file, operating in parallel, so a single cluster can support multiple logical queues to accelerate work output.

HPCC Systems also provides additional dataflow parallelism capabilities thanks to its programming language, ECL. ECL was specifically designed for data query and manipulation, while most of the languages used to program Spark (Scala or Python, for example) were not.

ECL code is declarative ("Here's the data analysis I want. Now find me the fastest way to get it.") instead of procedural ("Here's the data analysis I want, and I'll tell you how to go about getting it."). This means programmers only need to declare what their data-related needs are and the ECL compiler will optimize the sequence of operations and data manipulations in a parallel and distributed fashion. In practice, this means that just a few lines of ECL code can deliver what SQL or other languages would require hundreds of lines to accomplish. ECL can do this because after ECL code is compiled, it becomes a shared object sent to all nodes in the cluster for execution. Combining this capability with ECL's data locality awareness helps keep as much data processing on the local node as possible to minimize communication across nodes that delay the workflow.

After a programmer writes their ECL code, it is submitted to the compiler via a job container referred to as workunit. The ECL compiler server and ECL agent server then work together to control the workunit submission to the execution queues. A detailed look at how this process works can illustrate just how convenient and time-saving it can be for the compiler to manage the workload required by the code.

First, the compiler server parses the ECL code to determine what the result must be, then generates optimized C++ code to achieve that result, which is then compiled into machine code. During this process, the compiler will optimize the dataflow by properly ordering the sequence of operations, as well as spotting and eliminating data and operations that may not be relevant to the desired end result. This optimization is illustrated below.

ECL Code

```
1  rec := RECORD
2    STRING   firstname;
3    STRING   lastname;
4    UNSIGNED age;
5  END;
6
7  ds := DATASET([{'Wendy','Brown',34},
8                 {'Jimmy','Johnson',37},
9                 {'Nathan','Brown',45},
10                {'Carly','Brown',56},
11                {'Alfie','Johnson',66}],rec);
12
13 OUTPUT(ds,NAMED('Result1')); // the result of this first OUTPUT() is computed and
14                             // provided to the user while the result of the second
15                             // OUTPUT() is being computed and generated in parallel
16
17 mysort1 := SORT(ds,firstname);  // this SORT() will be skipped by the compiler as
18                                 // it does not affect the query end result
19
20 mysort2 := SORT(mysort1,lastname); // this SORT() will only be executed after the data
21                                    // is filtered by the "age" value designated next as
22                                    // the compiler spotted that sorting a subset of data
23                                    // is the optimal sequence to generate the query result
24
25 myfilter := mysort2(age>50);
26
27 OUTPUT(myfilter,NAMED('Result2'));
```

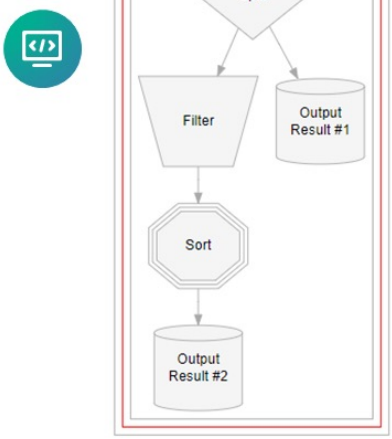


Figure 1: This ECL code sample and its execution graph illustrate the optimal sequence of dataflow operations generated by the compiler.

In this code example, a sample dataset composed of records containing "firstname", "lastname", and "age" information of fictitious individuals is defined in lines 1 through 11. Then an OUTPUT action in line 13 produces a visualization of the dataset in a relational format. Next, in lines 17 and 20, two SORT functions are defined to order the dataset by "firstname" and "lastname," respectively. A filter is applied in line 25 to only retain records whose "age" field is greater than the value "50" and, finally, in line 27, a second OUTPUT action will display the resulting recordset. When this sample code is submitted, HPCC Systems generates the optimized dataflow illustrated on the graph at the right.

In the resulting graph, one can see that the two OUTPUT actions have been split into two different branches of the graph so independent results can be provided as soon as possible. More importantly, on the left branch, the filter operation was applied first so the subsequent SORT operation (usually a time-consuming operation in big data) only needs to deal with the relevant data subset. Also, the first SORT operation (by "firstname") was not included on the resulting graph as it isn't relevant to the end result requested in the second OUTPUT. Although programmatically simple, this example illustrates scenarios that may be hidden behind larger and more complex code bases that make it difficult, if not impossible, for the programmer to identify them without the assistance of the ECL compiler.

After compilation, the workunit is scheduled for execution on the cluster queues via the ECL agent server. During code execution, another layer of optimization can be present to heavy duty operations, like joining datasets based on a certain condition. HPCC Systems can shuffle data around in an attempt to create the best computing environment in terms of efficiency for the operation.

HPCC Systems architecture is more simplified and efficient for data representation. Data used for computation is stored in its raw format on physical disks and is generally only manipulated in memory (there can be spillover to the hard disk with larger datasets). Unlike Spark and its reliance on data manipulated as an RDD, HPCC Systems doesn't need to put raw data in a predetermined format. The raw data is usually left unchanged, but if changes to the schema are necessary, they are applied during read time or in memory. Once the data is loaded in memory, any changes to the original raw dataset during computation will result in a new recordset being created in memory or on the hard disk, according to the programmer's choice.

Recordsets store a full computational history and can be re-computed at any time either from the original source or from persisted files created in the data manipulation stage to save computation workload. The only requirement for this model is that data needs to be locally available on the cluster nodes.

Data partitioning with HPCC Systems can be logically implemented at the code level to redistribute the data in memory according to predefined criteria and/or to create subgroups of data to be processed independently and in parallel. Depending on the partitioning technique specified in the code, the number of partitions can reflect either the number of compute nodes or a specific feature of the data.

Data Delivery Capabilities

Spark was not originally designed to support real-time data queries or analysis of real-time data streams, though support for real-time is available via software packages from Spark ecosystem partners (for example, Apache Hawq or Apache Impala). If the datasets being queried are small enough, some Spark users find a traditional Relational Database Management System (RDBMS) offers a fast enough response time to be considered "real time" for their requirements.

The Spark ecosystem is making progress in adding support for real-time data streaming. For example, Databricks has created a real-time query scheduler. However, the scheduler can only support a finite number of data sources.

Data delivery is a core focus of HPCC Systems. The regular architectural design of an HPCC Systems cluster includes a Roxie cluster (a cluster specifically designed to service queries) as a separate collection of compute nodes dedicated to query execution. Built-in support for a query engine is unique to HPCC Systems; Spark and other big data platforms require a third-party add-on application.

Roxie supports concurrent queries stored in pre-compiled code and natively supports indexed distributed data. These features enable Roxie to respond to a query in milliseconds. The query logic is self-contained in pre-compiled ECL code that provides an interface where users can provide input values for their query at execution time. Each Roxie node holds a partition of the indexed data (along with its meta key) and a copy of the pre-compiled code, so each node in the cluster is capable of addressing query requests by using its copy of the pre-compiled code to request data from peer nodes.

A Roxie cluster relies on a server/agent design in which a query request is made through an API and passed to a Roxie server node. When a node accepts the query, it then messages all the needed workers for the data, consolidates the data, and returns the results to the calling process.

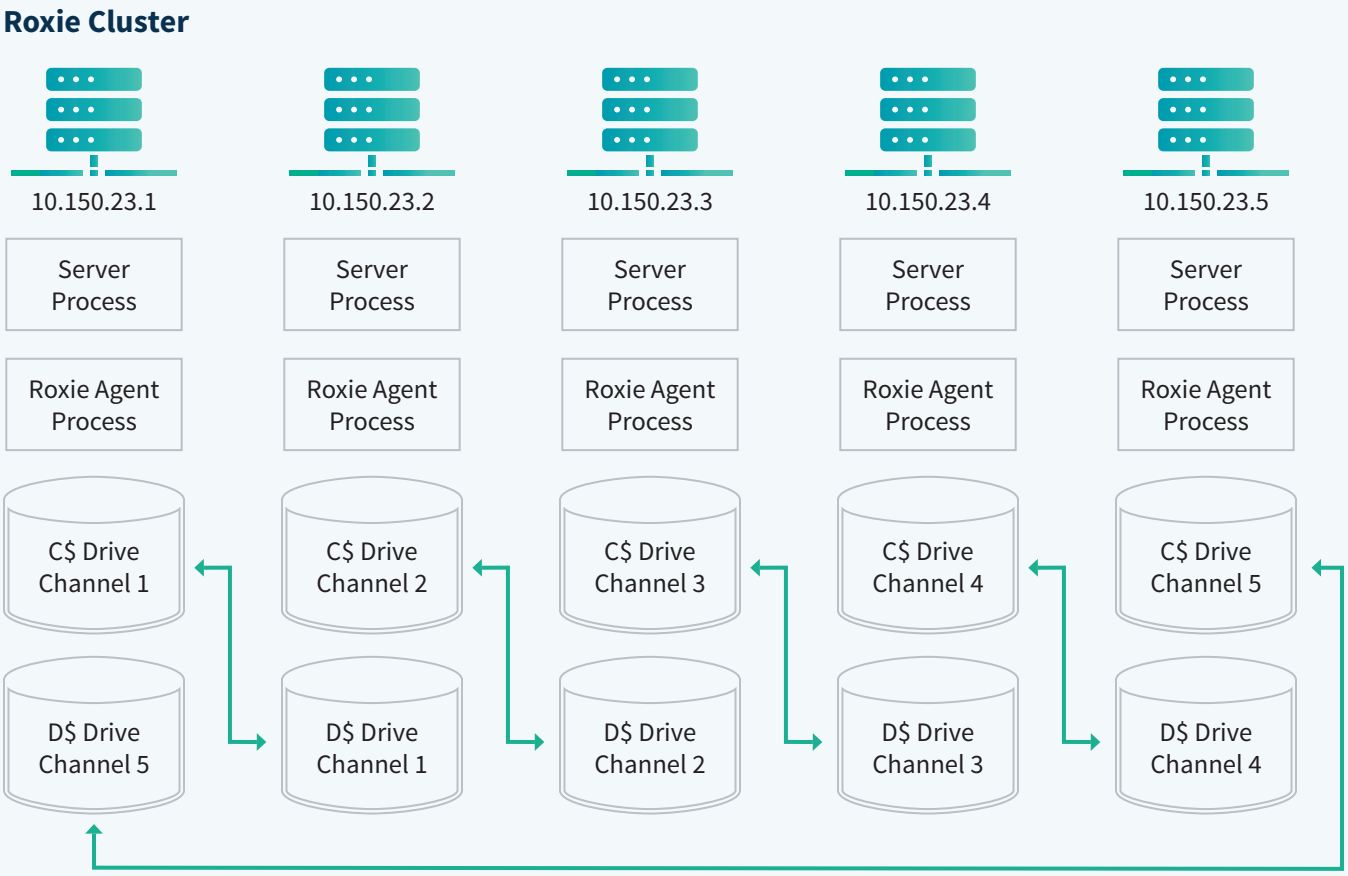
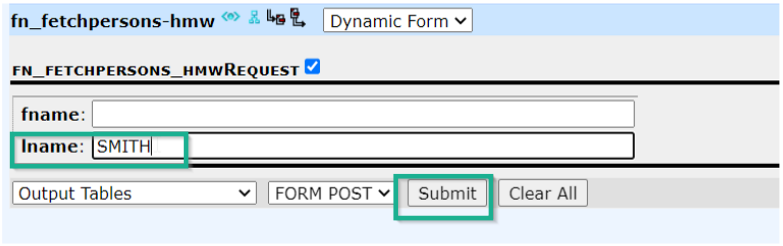


Figure 2: As illustrated in the diagram above, HPCC Systems Roxie cluster uses a server/agent design to manage concurrent data queries.

Roxie supports query requests via direct SOAP/REST calls, in batch mode or via specialized webservices provided by a middleware component called the Enterprise Services Platform (ESP). ESP manages and controls access to Roxie queries, allowing the use of query functions via more user-friendly interfaces like web forms.

Web Forms



Dataset: Result 1

	lastname	firstname	recid	middlename	namesuffix	filedate	bureaucode	gender	dependentcount	birthdate	streetaddress	city	state	zipcode
1	SMITH	DETELIN	596872	M		19861027	171	U	0	19650301	338 W 89TH ST APT 3R	ARLINGTON	MN	55307
2	SMITH	CELENIA	644126			19920319	199	F	0	19221201	2355 FOREST HILLS DR	TRANSFER	PA	16154
3	SMITH	TEH	623354	L	SR	19874081	238	M	0	19561201	18 CROSS RIDGE RD	EAST ORANGE	NJ	7017
4	SMITH	YAMROT	341777			20000810	168	F	0	19611214	280 W 25TH ST # C	FAYETTEVILLE	NC	28314
5	SMITH	GANIJA	44886	Z		19870909	13	F	0	19260301	2090 POTTS HILL RD	POMPTON PLAINS	NJ	7444
6	SMITH	CASIANO	643584	N		19871210	78	F	0	19620219	1754 PALISADE AVE	PERU	IL	61354
7	SMITH	RUEI	727071	S		19950921	252	M	0	19751201	43 HILLANDO DR	GLOUCESTER POIN	VA	23062
8	SMITH	ANNONAN	533969			19850821	376	F	0	19530108	134 E 17TH ST APT 63	JAMESON	MO	64647
9	SMITH	NAMIT	347861			19860401	24	M	0	19640928	221 CLERMONT AVE # 1	GLEN ELLYN	IL	60138
10	SMITH	MONTAKARN	277642	Q		19870309	13	M	0	19640929	45 MALLARD RD	DUNCANSVILLE	PA	16635
11	SMITH	VALDINA	609590	T		19940913	352	F	0	19730712	122 N BROAD ST	PITTSFIELD	NH	3263

Figure 3: HPCC Systems provides a simple, user-friendly interface for executing a data query on a Roxie cluster.

Support for delta lakes is another consideration developers should make when choosing a data lake platform. A delta lake is an open format storage layer providing reliability, security, and performance to an underlying data lake for both streaming and batch operations. Databricks has developed a solution for delta lake support on the Spark platform. HPCC Systems allows for use of a delta lake base in conjunction with Roxie queries to enable a hybrid Lambda design, which combines the capabilities of a transactional system (OLTP) with those of an analytical system (OLAP). The logic behind this is to use the delta lake to complement Roxie's query results with transactional data from external relational databases.

Many popular business intelligence tools (Tableau, for example) are compatible with Spark and use the platform's support for SQL to execute business intelligence queries. HPCC Systems was not originally designed with business intelligence in mind, but does support native data visualizations via a plug-in. For more advanced data visualization capabilities, Real BI markets a customizable data dashboard application that can quickly pull needed data from an HPCC Systems environment to test query concepts and confirm data accuracy.

Governance and Administration

For platform administration, Spark relies on third-party software libraries created and maintained by its ecosystem partners. While this gives Spark users flexibility in software choice, the initial setup and configuration of Spark with one of these libraries requires a lot of time and effort. Which administration method a Spark user should choose is highly dependent on the specifics of their deployment.

It's worth noting that, by default, Spark doesn't offer built-in solutions for security features like authentication and authorization. This is a serious issue in today's connected computing environments, where hackers work tirelessly to gain unauthorized access to data. To be secure, a data lake platform should be designed from the ground up with security in mind.

Administration is an integral part of the HPCC Systems stack. Management of data lineage, data copy/replication, data transfers, and publication between clusters is made easier since metadata from logical files are maintained within the cluster. By default, HPCC Systems provides the resources needed for day-to-day management of a cluster, including:

- ECL Agent component for controlling job submission
- ECL Watch service for monitoring workunits, cluster usage, components status, etc.
- Authentication and authorization management.
- Support for encryption of data while at rest and during transit
- Tools to support the maintenance and recovery of data
- Integration with open source monitoring tools such as Nagios and Ganglia

HPCC Systems implements security at a low level, rather than as a bolt-on feature. There are currently a few ways to use authentication with the HPCC Systems platform: simple htpasswd authentication, Lightweight Directory Access Protocol (LDAP), or another plug-in security method. The htpasswd authentication method is basic password authentication. It grants or denies access to a user based upon MD5 encrypted password authentication. LDAP authentication offers more features and options; it not only authenticates users, but also adds granularity to the authentication. When LDAP is configured, users need to be authenticated before accessing HPCC Systems services and those credentials are then used to authenticate any requests from the client side. LDAP allows you to control grouped access to features, functions, and files. Users should consider their system needs and decide which of these methods is appropriate for their environment.

As for user permissions and data security, their implementation and management is facilitated via Access Control Lists (ACLs) provided via the ECL Watch web browser interface. These ACL's allow system administrators to easily control user or group access to restricted data, workunits, or features of the HPCC Systems, such as ingesting data or accessing logs.

Access Control Lists

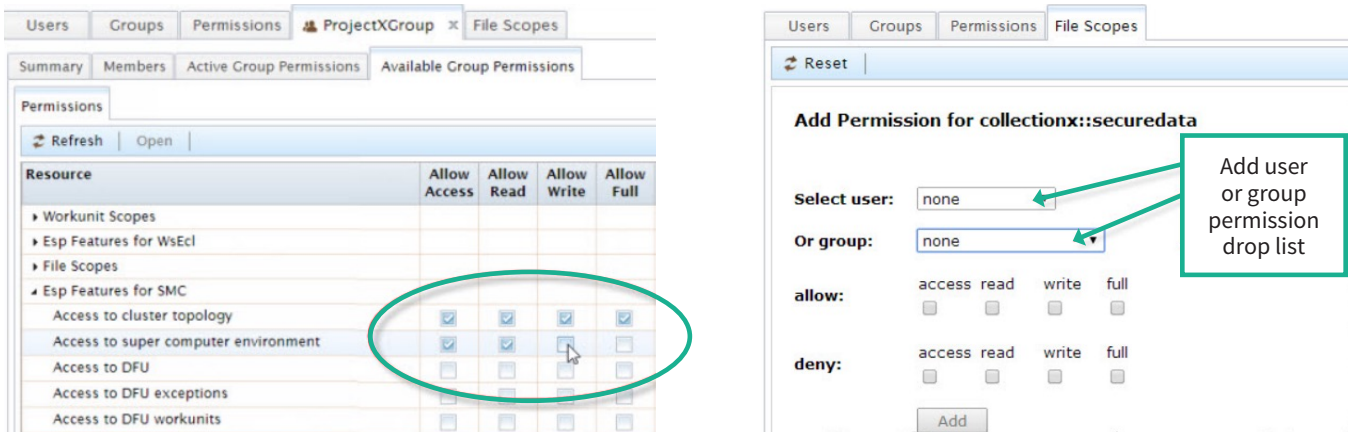


Figure 4: HPCC Systems allows facilitated control permissions via ACLs at the user and group level.

Spark lets users manage metadata via the Spark Catalog API. Datasource also developed a metadata storage management option for Spark that uses a delta lake to store metadata, maintain audit trails, and apply data schema enforcement.

On the HPCC Systems platform, basic metadata management is primarily done by an internal cluster component called DALI that stores cluster metadata in RAM. Recently, HPCC Systems has been developing a more specific solution for data management called Tombolo. For more details, please read the blog: Data Lake Curation and Automation with Tombolo. Or, please refer to our technical white paper, Data Lake Curation and Governance with Tombolo

As HPCC Systems was developed to be a full-service data lake platform and integrates many common administration tools, implementing HPCC Systems is a much easier experience and may require less IT support staff than a Spark implementation.

Cloud Native Support

Kubernetes (often abbreviated as "K8s") is an open source container orchestration system for automating software deployment, scaling, and management. K8s are widely used in many containerized cloud-based data applications and understanding how a platform supports K8s is important when choosing a data lake platform.

In terms of cloud native support, both platforms have been working on this capability in recent years. An experimental introduction of cloud native support for Spark started in 2018, but it was only declared generally available in 2021 with the release of Spark 3.1. HPCC Systems released its first cloud native version of the platform in early 2020, and currently v8.0.0 provides a complete cloud native version.

In terms of architectural design, the core approach in both cases is to convert the building blocks of each platform from a static bare metal or virtualized paradigm to a containerized microservices paradigm, which introduces significant changes in how components are configured, how and when they start, and where they store their data. So, although the overall code programming and job submission experience haven't dramatically changed, a complete redesign of the underlying infrastructure components was needed. For instance, in both cases, it was necessary to move away from local data storage assumptions at the node level and rely on centralized persistent volumes made accessible to those service components requiring access to the data, such as workers.

Spark uses spark-submit for job submission, which now can be used to submit a Spark application directly to a K8s cluster, but the driver and executor pod scheduling are handled by K8s. The submission mechanism works as follows: Spark creates a driver running within a K8s pod that in turn creates executors that also run within K8s pods.

Once the driver and the executors are connected, the application code is implemented. When completed, the executor pods terminate and are cleaned up, but the driver pod persists in a "completed" state in the Kubernetes API until it's no longer needed and deleted. Note that in the completed state, the driver pod does not use any computational or memory resources.

Similarly, for HPCC Systems, once a job is submitted, a stub component requiring minimal resources will identify the workunit and launch an independent K8s job to perform the actual compiling, including all data flow optimizations. From there, the ECL Agent component launches a K8s job for simple data manipulations and the Thor stub starts a Thor cluster only when needed. Thanks to this design, not only does the capacity of the system automatically scale up to use as many pods as necessary to handle the load, it scales down to use minimal resources (as little as a fraction of a single node) while waiting for jobs to be submitted.

So as one can see, the job management control and data flow optimization capabilities that were a core advantage for HPCC Systems in on-premises deployments are maintained in the cloud native world, with the added benefits of better managing your costs. HPCC Systems has also introduced several new informational features aligned to the cloud native paradigm, allowing users to be more knowledgeable about the costs associated with their jobs.

Cloud Native Support

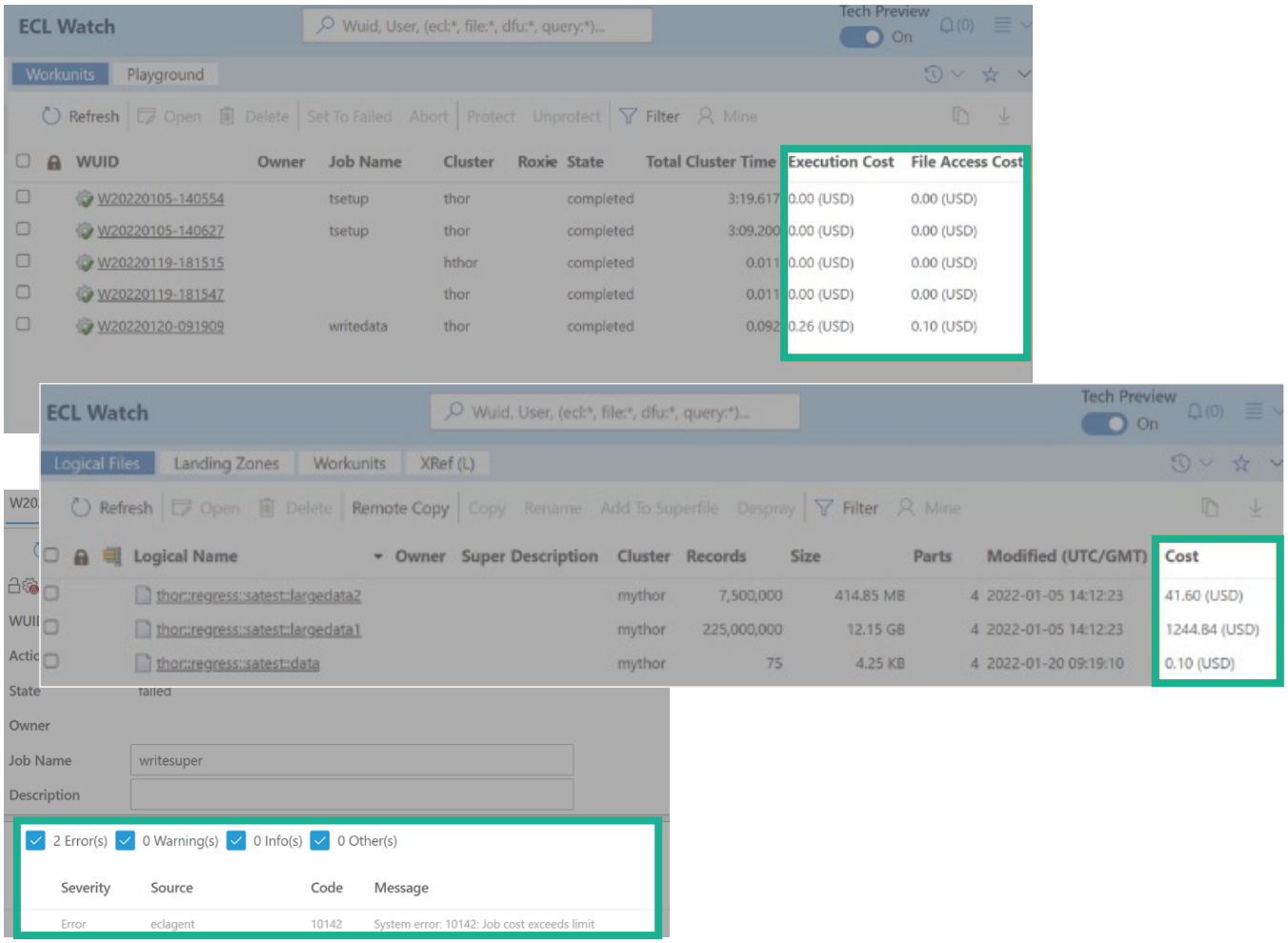


Figure 5: New HPCC Systems features introduced with cloud native support include cost information for workunit execution and file access/storage of datasets.

The current helm charts made available to the open source community still allow new users to deploy a complete end-to-end HPCC Systems in K8s within a few hours (or in minutes for users already familiar with cloud concepts). Recent contributions to the open source community also allow for full Infrastructure-as-a-Code (IaC) deployments using Terraform scripts that automate and abstract most of the effort required to build parts prior to the HPCC Systems platform deployment itself, such as building the K8s cluster, node pools, storage accounts, and more.

CONCLUSION

A complete Spark data analysis platform will require a resource-intensive setup process that can take weeks or months. This is primarily due to the need to complement Spark with many other third-party software and libraries in order to make it a complete data analysis platform. For example, Spark offers no built-in data storage support, so time and effort is required to source and configure one. Setup will also require someone familiar with Spark programming to avoid potential issues around memory dataset processing (refer to the Data Processing Architectural Comparison section earlier in the paper for more detail). If the data lake platform requires real-time data streaming and query support, more third-party software must be sourced and installed, creating further complexities for the deployment. And as much of Spark development comes from its ecosystem partners, users are reliant on licensed developers' product roadmaps to build the tools they need.

HPCC Systems is an all-in-one data lake platform that users can implement in just a few hours. HPCC Systems requires much less overhead for setup and administration as all components are part of the platform's technology stack. New platform deployment is further simplified by HPCC Systems open source licensing model. As HPCC Systems code is freely available to anyone who wants it, there is ample support documentation for programmers and system administrators to become familiar with the managerial aspects of the platform (i.e., fine tuning, resource configuration, security, data management, etc.). Since going open source in 2011, a global development community has shaped HPCC Systems into a mature and reliable platform. ECL training courses are free, and the language is easy to learn.

HPCC Systems is also well prepared for migrating data lakes to the cloud, a major trend in data analysis. LexisNexis Risk Solutions, the company that originally inherited the development of the HPCC Systems stack, has a dedicated development team focused on solutions to simplify cloud migration (see Cloud Native section above).

Although the Spark project is open source and licensed under Apache, using a Spark implementation will likely require the user obtain licenses for any third-party software libraries their particular implementation needs, an expensive and time-consuming proposition. HPCC Systems is fully covered by its open source software license. A pure HPCC Systems implementation gives users access to a

complete, fully functional, end-to-end data lake platform without the need for additional third-party software licenses. Therefore, while a specific cost-benefit analysis of whether Spark or HPCC Systems is a better fit is highly dependent on the specifics of each implementation, in general the total cost of ownership for HPCC Systems is lower.

Given these facts, it is easier for a smaller, less experienced IT team to setup and maintain a data lake platform based on HPCC Systems. While Spark does have some advantages over HPCC Systems in regard to programming language support via integrated API's, most users will likely find that benefit outweighed by the additional work involved in sourcing, procuring, installing, and managing the host of third-party software solutions Spark must use to provide end-to-end data lake management capabilities. Spark does have a large user base, and some organizations may have a pre-existing investment in Spark infrastructure that they need to keep active. Fortunately, HPCC Systems and Spark are interoperable.

In summary, the HPCC Systems data lake platform provides significant benefits over Spark in terms of completeness, level of integration, software costs, and initial deployment time; ultimately resulting in reduced overall total cost of ownership and time to market for complex data products. These facts, plus its support from a global network of open source developers, makes HPCC Systems a compelling choice for any organization's data lake needs.

For more information, call 877.316.9669 or visit www.hpccsystems.com

HPCC™ SYSTEMS

LexisNexis® RISK SOLUTIONS

About HPCC Systems®

HPCC Systems® from LexisNexis® Risk Solutions is a proven, comprehensive, dedicated data lake platform that makes combining different types of data easier and faster than competing platforms — even data stored in massive, mixed schema data lakes . It's also open source, free to use, and easy to learn. You can acquire, enrich, deliver and curate information faster using HPCC Systems — and the automation of Kubernetes in our cloud native architecture makes it easy to set-up, manage and scale your data to save time and money, now and in the future. HPCC Systems offers a consistent data-centric programming language, two processing platforms and a single, complete end-to-end architecture for efficient processing. To learn more, visit us at hpccsystems.com.

About LexisNexis® Risk Solutions

LexisNexis® Risk Solutions harnesses the power of data and advanced analytics to provide insights that help businesses and governmental entities reduce risk and improve decisions to benefit people around the globe. We provide data and technology solutions for a wide range of industries including insurance, financial services, healthcare and government. Headquartered in metro Atlanta, Georgia, we have offices throughout the world and are part of RELX (LSE: REL/NYSE: RELX), a global provider of information-based analytics and decision tools for professional and business customers. For more information, please visit www.risk.lexisnexis.com and www.relx.com.