

HPCC / Conector do Spark

Equipe de documentação de Boca Raton



HPCC / Conector do Spark

Equipe de documentação de Boca Raton

Copyright © 2024 HPCC Systems®. All rights reserved

Sua opinião e comentários sobre este documento são muito bem-vindos e podem ser enviados por e-mail para <>

<docfeedback@hpccsystems.com>

Inclua a frase **Feedback sobre documentação** na linha de assunto e indique o nome do documento, o número das páginas e número da versão atual no corpo da mensagem.

LexisNexis e o logotipo Knowledge Burst são marcas comerciais registradas da Reed Elsevier Properties Inc., usadas sob licença.

HPCC Systems® é uma marca registrada da LexisNexis Risk Data Management Inc.

Os demais produtos, logotipos e serviços podem ser marcas comerciais ou registradas de suas respectivas empresas.

Todos os nomes e dados de exemplo usados neste manual são fictícios. Qualquer semelhança com pessoas reais, vivas ou mortas, é mera coincidência.

2024 Version 9.8.94-1

O conector Spark HPCC Systems	4
Visão geral	4
Classes Primárias	6
Classes Adicionais de Interesse	9

O conector Spark HPCC Systems

Visão geral

O conector distribuído Spark-HPCCSystems é uma biblioteca Java que facilita o acesso de um cluster do Spark aos dados armazenados em um cluster do HPCC Systems. A biblioteca de conectores emprega o recurso de leitura de arquivos remotos padrão do HPCC Systems para ler dados de conjuntos de datasets sequenciais ou indexados do HPCC.

Os dados em um cluster HPCC são particionados horizontalmente, com dados em cada nó do cluster. Depois de configurados, os dados do HPCC estão disponíveis para acesso em paralelo pelo cluster do Spark.

No repositório do GitHub (<https://github.com/hpcc-systems/Spark-HPCC>) você pode encontrar o código-fonte e exemplos. Existem vários artefatos na pasta `DataAccess/src/main/java` de interesse primário. A classe `org.hpccsystems.spark.HpccFile` é a fachada de um arquivo em um cluster HPCC. O `org.hpccsystems.spark.HpccRDD` é um dataset distribuído resiliente derivado dos dados no cluster HPCC e é criado pelo método `org.hpccsystems.spark.HpccFile.getRDD(...)`. A classe `HpccFile` suporta o carregamento de dados para construir um objeto `Dataset <Row>` para a interface Spark. Isso primeiro carregará os dados em um RDD `<Row>` e, em seguida, converterá esse RDD em um `DataSet <Row>` por meio dos mecanismos internos do Spark.

Existem vários artefatos adicionais de algum interesse. A classe `org.hpccsystems.spark.ColumnPruner` é fornecida para permitir a recuperação somente das colunas de interesse do cluster HPCC. O artefato `targetCluster` permite especificar o cluster HPCC no qual o arquivo de destino existe. A classe `org.hpccsystems.spark.thor.FileFilter` é fornecida para facilitar a filtragem de registros de interesse do cluster HPCC.

O repositório git inclui dois exemplos na pasta `Examples/src/main/scala` folder. Os exemplos (`org.hpccsystems.spark_examples.DataFrame_Iris_LR` e `org.hpccsystems.spark_examples.Iris_LR`) são Scala Objects com uma função `main()`. Ambos os exemplos usam o dataset clássico da Iris. O dataset pode ser obtido de uma variedade de fontes, incluindo o repositório HPCC-Systems/ecl-ml. `IrisDs.ecl` (pode ser encontrado na pasta `ML/Tests/Explanatory`: <https://github.com/hpcc-systems/Spark-HPCC/blob/master/Examples/src/main/ecl/IrisDS.ecl>) pode ser executado para gerar o dataset Iris no HPCC. Um passo a passo dos exemplos é fornecido na seção Exemplos.

O conector distribuído Spark-HPCCSystems também suporta o PySpark. Ele usa as mesmas classes/API que o Java.



Como é comum na comunicação do cliente Java por TLS, os conectores Spark-HPCC direcionados a um cluster HPCC por TLS precisarão importar os certificados apropriados para o keystore Java local.

*Uma maneira de fazer isso é usar o keytool fornecido com as instalações Java. Consulte a documentação do keytool para uso.

Integração Spark

O plug-in Spark integrado ao HPCC não é mais compatível a partir da versão 9.0.0 em favor de clusters Spark autônomos gerenciados pelo usuário vinculados à plataforma HPCC usando o conector Spark-HPCC.

Considerações Especiais

Estouro de valor não assinado

Java não suporta um tipo de inteiro não assinado, portanto, a leitura de valores UNSIGNED8 dos dados do HPCC pode causar um estouro de inteiro em Java. Os valores de UNSIGNED8 são frequentemente usados como identificadores exclusivos em datasets, caso em que o overflow seria aceitável, pois o valor do transbordamento ainda será exclusivo.

O conector Spark-HPCC permite que os valores não assinados sejam excedidos em Java e não relatará uma exceção. O chamador é responsável por interpretar o valor com base no sinalizador recef **isunsigned**.

Classes Primárias

A classe *HpccFile* e as classes *HpccRDD* são discutidas em mais detalhes abaixo. Essas são as classes principais usadas para acessar dados de um cluster HPCC. A classe *HpccFile* suporta o carregamento de dados para construir um objeto *Dataset <Row>* para a interface Spark. Isso primeiro carregará os dados em um RDD *<Row>* e, em seguida, converterá esse RDD em um *DataSet <Row>* por meio dos mecanismos internos do Spark.

A classe *org.hpccsystems.spark.HpccFile* possui vários construtores. Todos os construtores recebem informações sobre o cluster e o nome do dataset de interesse. As classes JAPI WS-Client são usadas para acessar informações detalhadas do arquivo. Uma definição usada para selecionar as colunas a serem retornadas e uma definição para selecionar as linhas a serem retornadas também podem ser fornecidas. Eles são discutidos na seção *Classes Adicionais de Interesse* abaixo. A classe tem dois métodos de interesse primário: o método *getRDD(...)* e o método *getDataframe(...)*, que são ilustrados na seção *Exemplo*.

O método *getRecordDefinition()* da classe *HpccFile* pode ser usado para recuperar uma definição do arquivo. O método *getFileParts()* pode ser usado para ver como o arquivo é particionado no cluster HPCC. Esses métodos retornam as mesmas informações que podem ser encontradas na aba DEF da página de detalhes do dataset do ECL Watch e na aba PARTS respectivamente.

A classe *org.hpccsystems.spark.HpccRDD* estende a classe de modelo *RDD<Record>*. A classe emprega a *org.hpccsystems.spark.HpccPart* para as partições Spark. A classe *org.hpccsystems.spark.Record* é usada como o contêiner para os campos do cluster HPCC. A classe *Record* pode criar uma instância *Row* com um esquema.

Os objetos de partição *HpccRDD* *HpccPart* leem cada um blocos de dados do cluster HPCC independentemente uns dos outros. A leitura inicial busca o primeiro bloco de dados, solicita o segundo bloco de dados e retorna o primeiro registro. Quando o bloco estiver esgotado, o próximo bloco deverá estar disponível no soquete e uma nova solicitação de leitura será emitida.

O *HpccFileWriter* é outra classe primária usada para gravar dados em um cluster HPCC. Tem um único construtor com a seguinte assinatura:

```
public HpccFileWriter(String connectionString, String user, String pass)
throws Exception {
```

O primeiro parâmetro *connectionString* contém as mesmas informações que o *HpccFile*. Deve estar no seguinte formato: {http|https}://{ECLWATCHHOST}:{ECLWATCHPORT}

O construtor tentará se conectar ao HPCC. Esta conexão será então usada para quaisquer chamadas subsequentes para o *saveToHPCC*.

```
public long saveToHPCC(SparkContext sc, RDD<Row> scalarRDD, String clusterName,
String fileName) throws Exception {
```

O método *saveToHPCC* suporta apenas os tipos RDD<row>. Você pode precisar modificar sua representação de dados para usar essa funcionalidade. No entanto, essa representação de dados é usada pelo Spark SQL e pelo HPCC. Isso só é suportado gravando em uma configuração co-localizada. Assim, o Spark e o HPCC devem ser instalados nos mesmos nós. A leitura suporta apenas a leitura de dados de um cluster HPCC remoto.

O *clusterName*, conforme usado no caso acima, é o cluster desejado para gravar dados, por exemplo, no cluster Thor "mitor". Atualmente, há suporte apenas para gravação em clusters do Thor. A gravação em um cluster Roxie não é suportada e retornará uma exceção. O nome do arquivo usado no exemplo acima está no formato HPCC, por exemplo: "~example::text".

Internamente, o método `saveToHPCC` gerará múltiplos jobs do Spark. Atualmente, isso gera dois jobs. O primeiro job mapeia o local das partições no cluster do Spark para fornecer essas informações ao HPCC. O segundo job faz a gravação real dos arquivos. Há também algumas chamadas internamente ao ESP para lidar com coisas como iniciar o processo de gravação usando `DFUCreateFile` e publicar o arquivo depois de ter sido escrito chamando `DFUPublishFile`.

Using the Spark Datasource API to Read and Write

Example Python code:

```
# Connect to HPCC and read a file
df = spark.read.load(format="hpcc",
                     host="127.0.0.1:8010",
                     password="",
                     username="",
                     limitPerFilePart=100,
                     # Limit the number of rows to read from each file part
                     projectList="field1, field2, field3.childField1",
                     # Comma separated list of columns to read
                     fileAccessTimeout=240,
                     path="example::file")

# Write the file back to HPCC
df.write.save(format="hpcc",
              mode="overwrite",
              # Left blank or not specified results in an error if the file exists
              host="127.0.0.1:8010",
              password="",
              username="",
              cluster="mythor",
              path="example::file")
```

Exemplo de código Scala:

```
// Read a file from HPCC
val dataframe = spark.read.format("hpcc")
    .option("host", "127.0.0.1:8010")
    .option("password", "")
    .option("username", "")
    .option("limitPerFilePart", 100)
    .option("fileAccessTimeout", 240)
    .option("projectList", "field1, field2, field3.childField")
    .load("example::file")

// Write the dataset back
dataframe.write.mode("overwrite")
    .format("hpcc")
    .option("host", "127.0.0.1:8010")
    .option("password", "")
    .option("username", "")
    .option("cluster", "mythor")
    .save("example::file")
```

Exemplo de código R:

```
df <- read.df(source = "hpcc",
             host = "127.0.0.1:8010",
             path = "example::file",
             password = "",
             username = "",
             limitPerFilePart = 100,
             fileAccessTimeout = 240,
             projectList = "field1, field2, field3.childField")

write.df(df, source = "hpcc",
        host = "127.0.0.1:8010",
        cluster = "mythor",
        path = "example::file",
        mode = "overwrite",
        password = "",
        username = "",
        fileAccessTimeout = 240)
```


Classes Adicionais de Interesse

As principais classes de interesse para esta seção são a remoção de colunas e a filtragem de arquivos. Além disso, há uma classe auxiliar para remapear informações de IP quando necessário, e isso também é discutido abaixo.

As informações de seleção da coluna são fornecidas como uma string para o objeto *org.hpccsystems.spark.ColumnPruner*. A string é uma lista de nomes de campos separados por vírgulas. Um campo de interesse pode conter um conjunto de dados de linha ou filho e a notação de nome pontilhada é usada para oferecer suporte à seleção de campos filho individuais. O *ColumnPruner* analisa a cadeia em uma instância da classe *TargetColumn* raiz que contém as colunas de destino de nível superior. Um *TargetColumn* pode ser um campo simples ou pode ser um conjunto de dados filho e, portanto, ser um objeto raiz para o layout do registro filho.

O filtro de linha é implementado na classe *org.hpccsystems.spark.thor.FileFilter*. Uma instância de *FileFilter* é restrita a partir de uma matriz de objetos *org.hpccsystems.spark.thor.FieldFilter*. Cada instância de *FieldFilter* é composta de um nome de campo (em notação pontuada para nomes compostos) e uma matriz de objetos *org.hpccsystems.spark.thor.FieldFilterRange*. Cada instância de *FieldFilterRange* pode ser um intervalo aberto, ou fechado ou um valor único. O registro é selecionado quando pelo menos um *FieldFilterRange* corresponde para cada uma das instâncias do *FieldFilter* na matriz.

Os valores *FieldFilterRange* podem ser cadeias ou números. Existem métodos fornecidos para construir os seguintes testes de intervalo: igual, não igual, menor que, menor que ou igual a, maior que, e maior que ou igual a. Além disso, um teste de inclusão de conjunto é suportado para cadeias de caracteres. Se o arquivo for um índice, os campos de filtro, que são campos-chave, são utilizados para uma pesquisa de índice. Qualquer campo de filtro não mencionado é tratado como desconhecido.

A arquitetura de implantação usual para os Clusters HPCC consiste em uma coleção de nós em uma rede. As informações de gerenciamento de arquivos incluem os endereços IP dos nós que contêm as partições do arquivo. As classes do conector Spark-HPCC usam esses endereços IP para estabelecer conexões de soquete para a leitura remota. Um cluster HPCC pode ser implantado como um cluster virtual com endereços IP privados. Isso funciona para os componentes do cluster porque eles estão todos na mesma LAN privada. No entanto, os nós do cluster Spark podem não estar na mesma LAN. Nesse caso, a classe *org.hpccsystems.spark.RemapInfo* é usada para definir as informações necessárias para alterar o endereçamento. Existem duas opções que podem ser usadas. A primeira opção é que cada nó de trabalho do Thor pode receber um IP visível para o cluster do Spark. Esses endereços devem ser um intervalo contíguo. A segunda opção é atribuir um IP e um intervalo contíguo de números de porta. O objeto *RemapInfo* é fornecido como um parâmetro.

Fornecemos alguns exemplos de utilização de um ambiente Spark. Os exemplos fornecidos dependem do shell Spark..

Você pode encontrar exemplos no repositório do Github:

<https://github.com/hpcc-systems/Spark-HPCC/tree/master/Examples>