A Distributed Catalog for the Borealis Stream Processing Engine

Bradley Berg

May 31, 2006

Abstract

The Borealis stream processing engine distributes a network of queries over multiple computers. A Distributed Catalog was developed to capture the topology of the query diagram and it's deployment over the computer network.

A Borealis network may consist of many processing nodes distributed over a wide area network. Query optimizers can redistribute the processing load by changing the diagram topology or deployment. These changes can occur while data processing is in progress. The Distributed Catalog needs to be updated as changes occur.

This paper details how catalog information is disseminated and accessed throughout a Borealis network and components that may interact with the system. The contents of the Distributed Catalog need to be managed so that it remains coherent and changes are synchronized.

Contents

List of Figures

1 Introduction

Borealis [1] is a dataflow processing engine that distributes processing over multiple computers. Each Borealis node constitutes a single processing engine and executes a portion of the dataflow network. Nodes execute a collection of operators we call boxes.

Boxes are interconnected by dataflow streams that consist of a sequence of data tuples. Each tuple contains a set of data elements which are composed of numeric values, character strings or timestamps. The layout of the data elements in a tuple is called a schema. An individual box reads data from it's input streams, transforms the data and generates output streams. For example a filter box may read a sequence of tuple containing integers and then propagate positive integers to it's output stream.

The catalog is a data structure describing the topology of the dataflow network. It contains definitions of streams, tuples, boxes and the interconnections between these components. As a stream processing application runs the topology of the network may change. The application might add, delete, start, or stop components. Optimizers automatically manage the processing load by modifying the network. For example boxes may be move from a heavily loaded processor to one that is lightly loaded.

Changes to the dataflow network topology must be deployed to the affected Borealis nodes. The contents of the catalog must remain consistent across multiple computers and changes distributed in a timely fashion to allow the network to scale to many processors. This paper describes the Distributed Catalog for Borealis and how it achieves these goals.

2 The Distributed Catalog

Each Borealis node has a Local Catalog containing the slice of the topology deployed on the individual node. That is, boxes, streams and schemas that are located on a Borealis node are declared in the Local Catalog for that node. Local Catalogs are the primary reference for catalog information and collectively form a complete distributed catalog. The Local Catalogs contain the most up to date processing state of the complete Borealis system. Individual Borealis nodes are able to independently perform the operations they need without accessing external catalog information.

A Global Catalog describing the complete Borealis network is also maintained by a program we refer to as the Head. An application program sends topology declarations and modifications as XML to the Head. The Head validates the application XML against the contents of the global catalog to maintain consistency over the dataflow topology. Once accepted, the Head deploys the updates to the affected Borealis nodes so they may implement local changes to the network; updating their local catalog in the process.

The Head can operate persistently over the life of the application or it can simply deploy a static topology and terminate. A persistent Head enables and tracks dynamic modifications of the network topology. Applications pass XML as strings or files to the persistent Head using Remote Procedure Calls. Static topologies are passed to the Head in files listed in program arguments.

2.1 Updating the Catalog From the Head

Applications send two sets of XML to the head; one representing Borealis topology declarations and the other specifying how the topology is to be deployed over multiple Borealis nodes. The formal definition of the XML is in the Document Type Definition file (borealis.dtd) for Borealis XML. It can be referenced by Application XML files to improve syntax checking. An informal description of the Application XML is in the Borealis Application Programmer's Guide [2].

The following example illustrates how to define and deploy a simple Borealis network. First the Application XML in Figure 1 defining the network topology is sent by the application program to the Head. The Head then validates the declarations and adds them to the Global Catalog. Next the application sends the Deployment XML in Figure 2 to the Head describing how the network is to be deployed. After receiving deployment XML the Head sends declarations to the appropriate Borealis nodes.

In this case a network with only one operator (mybox) is deployed to a single Borealis node running on the same computer as the Head, localhost. The endpoints associated with the node and the input and output streams specify IP socket ports used to transmit data between the application and Borealis nodes.

```
<?xml version="1.0"?>
<!DOCTYPE borealis SYSTEM
          "http://www.cs.brown.edu/research/borealis/borealis.dtd" >

<borealis>
    <input   stream="Packet"     schema="PacketTuple"    />
    <output  stream="Aggregate"  schema="AggregateTuple" />

    <schema name="PacketTuple">
        <field  name="time"        type="int" />
        <field  name="protocol"    type="string"  size="4" />
    </schema>

    <schema name="AggregateTuple">
        <field  name="interval"    type="int" />
        <field  name="count"       type="int" />
    </schema>

    <box name="mybox"   type="aggregate" >
        <in   stream="Packet"    />
        <out  stream="Aggregate" />

        <parameter   name="aggregate-function.0"  value="count()"  />
        <parameter   name="window-size-by"        value="VALUES"   />
        <parameter   name="window-size"           value="1"        />
        <parameter   name="advance"               value="1"        />
        <parameter   name="order-by"              value="FIELD"    />
        <parameter   name="order-on-field"        value="time"     />
    </box>
</borealis>
```

Figure 1: Application XML to Define a Simple Borealis Topology

```
<?xml version="1.0"?>
<!DOCTYPE borealis SYSTEM
          "http://www.cs.brown.edu/research/borealis/borealis.dtd" >

<deploy>
    <publish    stream="Packet"     endpoint="localhost:15000" />
    <subscribe  stream="Aggregate"  endpoint="localhost:25000" />

    <node   endpoint="localhost:15000"    query="mybox" />
</deploy>
```

Figure 2: Deployment XML for the toplogy in Figure 1

- Borealis XML uses endpoints to assign communication channels to streams, Update XML, and Remote Procedure Calls. The endpoint is assigned to the component on the receiving end of the channel. In this example the Aggregate stream is received by an application using the channel "localhost:25000"; where localhost refers to the machine running the application and 25000 is a channel port number.

When the Head deploys elements to specific Borealis nodes, it breaks up the XML received from the application into individual elements and propagates each element to the appropriate Borealis node. The XML sent from the Head to the Borealis nodes is called Update XML. It is similar to the original XML element except that some elements have more attributes and additional elements are added to support inter-node transformations.

The architecture of the Distributed Catalog allows Borealis systems to scale to large numbers of nodes spread throughout a wide area network. It is important to note that the Head passes updates to Borealis nodes, but that the nodes do not need to reference the Global Catalog in the Head. Instead nodes rely on their Local Catalog.

XML is passed through the Head and then only the portions required by a node are passed on. The communication between the Head and nodes is the bare minimum required to update each node. Likewise the Head only needs to propagate topology updates to affected nodes. Unaffected nodes can continue processing without interruption.

Also note that the communication paths are forward stores. Reads involve a request, wait for response and receive sequence. They introduce unacceptable latencies and must be serialized or use locks to synchronize multiple messages. Nodes never need to wait to access the Global Catalog.

2.2 The Regional Catalog

Some components such as load optimizers and monitoring tools may need catalog information spanning several Borealis nodes. A region contains an arbitrary set of distributed Borealis nodes. The set of nodes included in a region can be determined by the application or by system tools. Regions may overlap and not all nodes need to be included in any region. In general nodes are assigned to a region based on some criteria such as connectedness, network latency or physical locality.

The regional catalogs and the global catalog are updated by reflecting changes to local catalogs using Update XML. Each Borealis node contains a list of endpoints that regional components use to receive Update XML. The list within a particular Borealis node has only regional components that include the node in their regions. An application or regional component can specify the set of nodes included in a region using the XML region element.

The Head is always included in the list as it includes all Borealis nodes in it's region. However updates submitted by the Head are not reflected back to the Head. Only changes instigated from distributed Optimizers and High Availability components are reflected to the head.

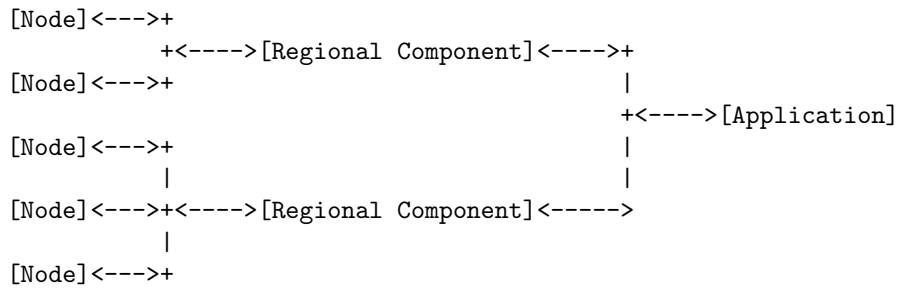


Figure 3: Regional Update XML and Data Flows

A Regional Component is a program that interacts with the nodes in a region. Regional components usually incorporate a Regional Catalog which covers the Local Catalogs of the nodes within the region. The Regional Catalog is updated as Borealis nodes reflect changes made to their Local Catalogs. When a node receives Update XML from the Head it reflects a copy of the XML to all regions to which the node belongs. Consequently the Regional Catalog may be out of date due to latency in reflecting updates. Regional components need to take this into consideration.

Note that if desired, Regional Components can propagate information to other programs spanning multiple regions. This hierarchical architecture allows data to propagate and merged for centralized processing; avoiding centralized access to Borealis nodes.

An application can send a request for a program to receive updates from a Borealis nodes by sending XML to the Head that includes:

```
<deploy>
    <--!  Launch a program and add nodes to it's region.
     -->
    <program  [name={program}]  [file={executable}]   node={region}] >
        <region  [endpoint={region}  node={node endpoint} />

        <region  node={node endpoint} />
                  :
    </program>

    <region  [name={region name}]  [endpoint={region}]
              node={node endpoint} />

    <region  node={node endpoint} />
                  :
</deploy>
```

Figure 4: XML Defining Access to Regions

The reflection mechanism used to update the regional catalog by the nodes in the region uses forward store messaging to avoid waiting for reads. The updates will be slightly later than changes to local catalogs. Consequently the topology in the Regional Catalog may be different from the actual network topology. Regional Component design and implementation need to take this into account.

Design considerations for a Regional Component also need to include scalability issues. Since a Regional Component can communicate with several nodes, the communication bandwidth needs to be managed to enable scalability. Often developers will extend a Borealis node to incorporate a counterpart to a Regional Component to perform local operations. A local extension can filter or concentrate communications with a Regional Component to reduce communication bandwidth.

2.3 Accessing the Global Catalog

Similar to Regional Components, a Global Component spans all Borealis nodes. The catalog in a global component is a copy of the global catalog in the Head. It is updated by the Head when it receives new XML. While a Global Component enjoys a complete view of the topology it is least up to date.

A Global Component should not have direct contact with Borealis Nodes. If it does, communications should be highly selective. As with Regional Components, bandwidth analysis needs to be performed on any such communication to ensure the system is scalable. In the case of Global Components the constraints are even more restrictive. The hierarchical structure of Regional Components (see Figure 3) can be exploited to insulate Global Components from directed contact with nodes.

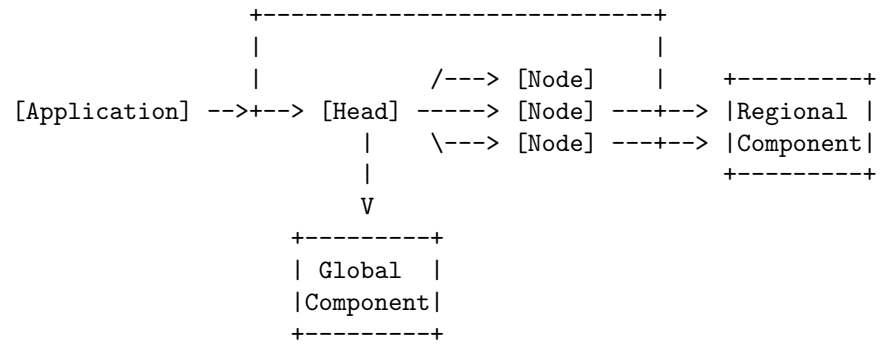


Figure 5: Flow of XML in a Borealis System

Tools that monitor the entire network or perform global network optimization will need a copy of the Global Catalog. Since the Head updates Global Components, ideally such tools should run on the same machine as the Head or one with a reliable low latency connection to the Head.

A Global Component can join a running Borealis system. In this case it needs to get an initial copy of the full catalog. This is done by sending a request to the Head for a complete copy of the Global Catalog. The Head responds by sending the catalog as a sequence of update XML to the Global Component at the designated endpoint. Loading the whole catalog is also useful for recovery after a Global Component temporarily fails, is shut down, or loses contact with the Head.

```
<global  endpoint={Global Component endpoint} />

<delete  global={Global Component endpoint} />
```

Figure 6: Connecting and Disconnecting a Global Component

3 Dynamic Network Topology Modification

Components such as optimizers and High Availability mechanisms can also instigate changes to the network topology from inside Borealis nodes or externally from a Regional or Global component. For example, an optimizer might want to offload processing from a heavily loaded computer to one that can handle additional load. When such a change occurs the affected Local Catalogs need to be changed and corresponding changes need to be propagated to the Global Catalog and affected Regional Catalogs.

Currently there are two transformations that can be applied directly to a Borealis node. A move-box operation relocates a box from one node to another. Moves are usually used to offload processing to another computer. An insert-box operation places a Filter Box with one input and one output along a stream. For example, an optimizer might insert a Drop Box that eliminates select tuples from the stream. This is done to shed load on a heavily loaded computer.

3.1 Moving a Box

When a box is moved from a source node to another target node, schemas for the streams connected to the box must also be moved. In the first stage of a move operation the source node sends Update XML containing the schemas and the box declaration to the target node.

In many cases the schemas are already declared in the target node's Local Catalog. In this case they are simply ignored as they are received by the target node. Once added to a system, schemas are never removed. Applications can not delete a schema and then redefine it. Not only would it be confusing to alter schema definitions, but persistent schemas simplify processing Update XML. For example, the same schema might be used by several streams, so the dependency analysis to remove it would be complex. Furthermore complete dependency analysis could not be done using only a node's Local Catalog.

Once the target node receives the declarations for the schemas and the box, it reflects the Update XML to the Regional Components for regions containing the target node. Concurrently the source node sends a 'move' XML element to the Head and to any Regional Components with regions containing the source node. The Head uses the move element to change the location of the box in the Global Catalog.

```
                  <schema ...>
[source]  -----> <schema ...>  ----> [target]  ----> [Region]
                  <box ...>                    ----> [Region]


                                                          ---> [Region]
[source] --> <move  box={box name}  node={target node} /> ---> [Head]
                                                          ---> [Region]
```

Figure 7: Update XML For Moving a Box

3.1.1 Propagating Box Moves To Regional Components

From the perspective of a Regional Component, on a move-box operation it may see a box declaration for a box moving into it's region, a move element for a box leaving it's region or both. When moving a box within a region a regional component will receive both and due to concurrency they will be unordered. The four possible scenarios for moving a box are:

- A box is moved into a region. The source node is out of the region and the target node is within the region. The target node reflects the schema and box declarations to the region and adds them to the Regional Catalog.
- A box is moved out of a region. The source node is within the region and the target node is out of the region. The source node sends the move element to the region (as well as to the Head). The region determines that the target node specified on the move is out of the region and the box is deleted. No schema definitions are deleted.
- A box is moved within a region and the move element is received before the box declaration. First the source node sends the move element to the region (as well as to the Head). The region determines that the target node specified on the move is within the region and changes it's location. Secondly, the box declaration is received. The regional component will ignore the declaration as the box is already declared in the region.
- A box is moved within a region and the box declaration is received before the move element. As before when the box declaration is received, the regional optimizer will ignore it as the box is already declared in the region. Then when the move is received the location of the box will be updated.

When multiple moves occur simultaneously, regional components may be receiving update XML out of order for the moves. However each move operation will be operating on different boxes, so there will not be a conflict.

3.1.2 Propagating Box Moves To the Head

When moving a box the source node will send a move element to the Head and that is all. The Head simply needs to modify its Global Catalog to change the location of the named box. Note however that there is a latency from the reference declaration in the Local Catalog to the declaration in the Global Catalog. If the Head needs to know the location of a moved box, its location in the Global Catalog may not yet be updated with the new location.

The Head may need to know the current location of a box if it has received a request to delete a query or dynamically create an input or output connection.

- If the location is checked after the move, there is no conflict. The location of the box in the Global Catalog matches the declaration in the target node's Local Catalog. The request is sent to the target node and completes successfully.
- When a request occurs during a move, the Head first sends the request to the source node. The request will fail as the box is no longer on the source node. When the Head receives the failed status it places the request into a queue of pending requests and waits. In a short while the move element sent by the source node will be received by the Head. The box location will be changed in the Global Catalog to the target node. The Head can then extract the pending request and resend it; this time to the updated target node.
- Although the window of opportunity is small, it is possible that the node could have been moved yet again. In this case the process repeats until the request succeeds. This is the Borealis version of Wack-A-Mole.

3.2 Inserting and Removing a Filter Box

A filter box with one In stream and one Out stream can be inserted along a select arc between two boxes. The schemas for the In and Out streams must be the same.

```
Before:    [Source Box] --existing--> [Sink box] --->

After:     [Source Box] --existing--> [Filter Box] --new--> [Sink box] -->
```

Streams have one source port and may fan out into several arcs connected to multiple box In ports. Filters can be inserted onto any one arc for a stream. In the simplest and most common case a stream will not fan out and the stream and arc will be synonymous. A single filter can not be inserted on all arcs or select arcs of a stream that fans out to multiple Sinks.

3.2.1 Inserting a Filter Box

A request to insert a filter box along a stream is made using a 'connect' Update XML element. Note that a new Out stream for the filter box is added to the topology as well.

```
<connect  box={Sink Box}  [port={zero based destination port}]
                      [upstream={0 | 1}] >

    <box name={box to insert}  type={box type} >
        <in   stream={existing stream} />
        <out  stream={new stream} />
        <parameter  name={parameter name}  value={parameter value} /> ...
   </Box>
</connect>
```

Figure 8: Request to Add a Filter

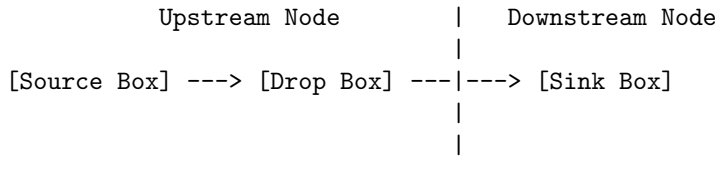
Note: A port index is the zero-based index of an In or Out stream on a box. Don't confuse it with a port number; which is used to select a communications channel in an endpoint.

Note that the port index for the In stream on the Sink Box determines a unique arc as there may be several for the stream. The box is inserted along a particular arc on the existing stream and not all arcs for the stream. The existing stream is disconnected from the Sink Box and replaces it with the new stream. The filter Box is then activated to resume processing Tuples.

In the simplest case a filter Box is inserted on the same node as the Sink Box. The upstream attribute is zero or is omitted. The Sink Box is suspended and the filter Box and the new stream are connected. When processing resumes tuples are seamlessly passed through the Filter box.

When inserting a box along a stream that runs between two nodes the box can be inserted on either of the connected nodes. Consider the case where we insert a Drop Box along a single stream connection. Since the bandwidth of the Drop's out stream is lower it's more efficient to place the Drop Box on the upstream node containing the Source Box. This will result in fewer tuples being send to the downstream node.

Inserting a box on an upstream node along a stream that runs between two nodes is initiated by setting the connect upstream attribute to one. The connect element is always sent to the downstream node containing the Sink Box. It is done this way because the originator of the requests needs to know the topology of the Sink Box in order to determine the in port index. The originator does not need to know anything about the Source Box.



A filter can not be inserted on an upstream node if the old stream fans out to multiple Sinks on the downstream node. If it could then both the old and new streams would need to be sent between the nodes. This would result in an increase in traffic and the resulting topology would be more complex.

After the downstream stream node receives the connect request it initiates a series of transactions to insert the box. First it modifies the connect request to replace the the box and port tags with it's endpoint and sends the modified request to the upstream node. The endpoint modification then informs the upstream node how to communicate with the downstream node.

```
<connect  node={downstream endpoint} >
    <box name={box to insert}  type={box type} >
        <in   stream={existing stream} />
        <out  stream={new stream} />
        <parameter  name={parameter name}  value={parameter value} /> ...
   </Box>
</connect>
```

Figure 9: Modified Request Sent to the Upstream Node

The two nodes then perform the transforms needed to insert and connect the box. First the upstream node adds the filter and the new stream. The publication of the old stream to the downstream node is removed and the new stream is published in its place. The downstream node then connects then replaces the old stream's connection to the designated In port on the Sink Box with the new stream. Again, when processing resumes tuples are seamlessly propagated though the filter box.

3.2.2 Removing a Filter Box

Removing a filter Box is initiated using a 'disconnect' Update XML element. The filter Box must have been inserted through the connect process. Only the new stream added for a connect needs to be designated. The relevant elements to be disconnected can be discerned from the new stream.

```
<disconnect  stream={new stream} />
```

On a single node the new stream will run from the filter Box to the Sink Box. In turn the filter Box In stream will be the old stream so it's identity is known as well. Once the local topology is known, the old stream is reconnected to the Sink Box and the filter Box and new stream are removed.

In a topology involving multiple nodes the disconnect element is sent to the downstream node containing the Sink Box. The communications endpoint for the upstream node can be extracted from the subscription to the new stream. The disconnect element is then forwarded to the upstream node; which uses it to locate the filter Box and the old stream. At this point the two nodes collaborate to republish the old stream and remove the filter Box and new stream.

References

[1] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The design of the Borealis stream processing engine. In *Proc. of the Second Biennial Conference on Innovative Data Systems Research (CIDR)*, Jan. 2005.

[2] Borealis application programmer's guide. `http://www.cs.brown.edu/research/borealis/public/publications/%borealis_application_guide.pdf`.