Agrios: A Hybrid Approach to Scalable Data Analysis Systems

Dissertation Proposal – 5 July 2012
Patrick Leyshock
advised by David Maier
The problem: scientists and engineers want to analyze large datasets …

- We need useful information from this data

- Data:
  - size often exceeds main memory
  - modeled as multidimensional arrays

- Traditional tools cannot do the job
  - Analytic systems do not perform well on large data
  - Database systems cannot perform sophisticated analytics

What to do?
Three strategies for tackling the problem

Make analytic systems work better with large datasets

Create database systems that perform sophisticated analytics

Combine an analytic system with a database: “let each do what it does best”
One example of a hybrid system: Agrios

- Integrates R and SciDB
- Middleware between the two systems
- Powerful analysis system and programming language
- Vectors and arrays are fundamental data objects
- High-level operations on data
- Open-source

```
res <- A %*% B
```
Array database management system
- Arrays are the fundamental data objects
- Low-level operations on arrays
- Scales well
- Open-source

```c
store(multiply(A,B), res);
```
My contributions include:

- Semantic mapping between the R language and SciDB's Array Functional Language
- Design of an automated, cost-based interaction model between R and SciDB, that reduces data movement
- Start-to-finish system implementation – Agrios – constructed using R and SciDB
- Test results quantifying the performance of this particular hybrid approach
Design principles – all things being equal, a good tool should not force users to:

• Learn new languages
• Learn new programming paradigms
• Maintain multiple versions of a functionally identical script
• Refactor scripts that already work well

These last two points relate to independence:

• user shouldn’t have to rewrite scripts if the location of the input data changes
• user shouldn’t have to rewrite scripts if the size of the input data changes

One way to achieve transparency is through automation.
Assumptions about hybrid systems

In all systems:

- scripts contain multiple operations
- faster is better

In hybrid systems:

- operations performed on both components
- data stored at both components

Data must be moved, and moving data isn't cheap. So, the amount of data moved is a performance driver for a hybrid system.
There are ways we can reduce the cost of data movement. We could improve:

- compression techniques
- partitioning schemes
- serialization formats
- etc.

Or, we could ensure we make good decisions about:

- what data to move, and
- when to move it
Decisions about data movement matter:

Vectors R and C are stored at B, and we need their product at A.

There are choices, and some choices are better than others.
What is a staging?

What are the decisions we need to make when trying to minimize data movement?

Different sides of the same coin:
- Where do we perform each operation in a script?
- Where do we move each data object in a script?

The answer to these questions is a *staging*. A staging states:
- the execution location for each operator in a script
- the destination location for each data object in a script
A sample expression:

\[
((A \%\% (\text{sum}(B) + C))[1:50,1:50])
\%\% ((D \%\% D) + F[1:100,1])[1:50,1]
\]
We need to fill in the question marks:

operation_1: R
operation_2: SciDB
operation_3: SciDB

Let’s walk through an example.
How do we determine a staging?

A simple expression:

Let’s assume, for ease of exposition, that we want the root operation performed at R.

How do we determine the staging?
Staging – the options

There are four different options to choose from:

Option 1: Do both subtrees at R
Option 2: Do the left subtree at R, the right subtree at SciDB
Option 3: Do the left subtree at SciDB, the right subtree at R
Option 4: Do both subtrees at SciDB
The four options require that different intermediate results be moved:

Option 1: Do both subtrees at R (move neither)
Option 2: Do the left subtree at R, the right subtree at SciDB (move results of right subtree)
Option 3: Do the left subtree at SciDB, the right subtree at R (move results of left subtree)
Option 4: Do both subtrees at SciDB (move results of both subtrees)
And each option has a different cost. Let’s cost one particular option.
We need to calculate the total cost of the plan.

What do we already know?

- Cost of left subtree at R: 100
- Cost of left subtree at SciDB: 1000
- Cost of right subtree at R: 2000
- Cost of right subtree at SciDB: 1
We need to calculate the total cost of the plan.

What do we already know?

- Cost of left subtree at R: 100
- Cost of left subtree at SciDB: 1000
- Cost of right subtree at R: 2000
- Cost of right subtree at SciDB: 1

The cost of the plan

= cost of left subtree at R (100)  
+ cost of right subtree at SciDB (1) 
+ cost of moving right subtree from SciDB to R
We calculate costs for each of the four options…
We calculate costs for each of the four options…

… and choose the best one.

We’ve used plan cost to determine the best staging.
Some possible stagings:

there are many more …
What if the location of the input data objects vary? We get different stagings, with different costs:

<table>
<thead>
<tr>
<th>scenario file</th>
<th>Initial object locations</th>
<th>Final result location</th>
<th>Staging</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>objects.size.1.R</td>
<td>S</td>
<td>R</td>
<td>S</td>
</tr>
<tr>
<td>objects.7.R</td>
<td>S</td>
<td>R</td>
<td>S</td>
</tr>
<tr>
<td>objects.8.R</td>
<td>S</td>
<td>R</td>
<td>S</td>
</tr>
<tr>
<td>objects.9.R</td>
<td>S</td>
<td>S</td>
<td>R</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>scenario file</th>
<th>Staged</th>
<th>All at R</th>
<th>All at SciDB</th>
<th>Analytics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># of internal transfer</td>
<td># of final elements transferred</td>
<td># of transfers</td>
<td># of elements transferred</td>
</tr>
<tr>
<td>objects.size.1.R</td>
<td>3</td>
<td>151</td>
<td>50</td>
<td>3</td>
</tr>
<tr>
<td>objects.7.R</td>
<td>3</td>
<td>151</td>
<td>50</td>
<td>3</td>
</tr>
<tr>
<td>objects.8.R</td>
<td>3</td>
<td>151</td>
<td>50</td>
<td>3</td>
</tr>
<tr>
<td>objects.9.R</td>
<td>3</td>
<td>151</td>
<td>50</td>
<td>3</td>
</tr>
</tbody>
</table>
What if the size of the inputs vary?
If the size of the inputs vary, we may get different stagings, with different costs:

Cost: 10,201 data elements

Cost: 20,000 data elements
Design principles – all things being equal, a good tool should not force users to:

- Learn new languages
- Learn new programming paradigms
- Maintain multiple versions of a functionally identical script
- Refactor scripts that already work well

These last two points relate to *independence*:

- user shouldn’t have to rewrite scripts if the location of the input data changes
- user shouldn’t have to rewrite scripts if the size of the input data changes

One way to achieve transparency is through automation.
We pursue three strategies for reducing data movement:

- identifying good stagings
- rewriting expressions -- enables additional stagings
- accumulating expressions – increases scope for rewriting and staging

The corresponding research questions are:

- How does staging impact data movement?
  - How much difference will staging make over other techniques?
  - How do we find good stagings?
- How does the rewriting of expressions impact possible stagings?
- How does the accumulation of expressions impact staging?
Select the best staging

What is the cheapest staging?
We need to calculate the total cost of the plan.

What do we already know?

- Cost of left subtree at R: 100
- Cost of left subtree at SciDB: 1000
- Cost of right subtree at R: 2000
- Cost of right subtree at SciDB: 1

The cost of the plan

= cost of left subtree at R (100)
  + cost of right subtree at SciDB (1)
  + cost of moving right subtree from SciDB to R
We need to calculate the total cost of the plan.

What do we already know?

- Cost of left subtree at R: 100
- Cost of left subtree at SciDB: 1000
- Cost of right subtree at R: 2000
- Cost of right subtree at SciDB: 1

The cost of the plan

= cost of left subtree at R (100)
  + cost of right subtree at SciDB (1)
  + cost of moving right subtree from SciDB to R
    (# of data elements * cost per data element)
Expression rewriting

A sample plan:

The best staging has a cost of 4000 data elements.
Rewrite the plan, using association:
Through expression rewriting, we’ve exposed new staging opportunities, decreasing the cost of the plan:

Best staging: 4000 data elements

Best staging: 2000 data elements

Once we begin rewriting expressions, the number of staging opportunities grows
Strategy 2 - execution

How do we rewrite expressions?
Agrios contains four major subsystems:

- Parser
- Accumulator
- Stager
- Executor
Agrios contains four major subsystems:

- Parser
- Accumulator
- Stager
- Executor

**Bonneville** is responsible for staging – it is an extension of the Columbia relational database optimizer.
Bonneville lets us define rewrite rules:

- Left-to-right associate
- Right-to-left associate
- Commute
- etc.
Bonneville lets us define rewrite rules:
  • Left-to-right associate
  • Right-to-left associate
  • Commute

These rewrite rules aren’t array-specific – they usually just let us *consolidate* data objects. These are “general” rules.
Other rules *are* array-specific, and work to reduce the amount of data moved. Consider this plan:
We can rewrite the plan by “pushing down” the subscript operation:

This can reduce the amount of data moved.
Now we have two different rule types:

**General rules:**
- Left-to-right associate
- Right-to-left associate
- Commute
- etc.

**Array-specific rules**
- Push subscript through elementwise addition
- Push subscript through matrix multiplication
- Distribute aggregate sum over elementwise addition
- Push aggregate sum through elementwise addition
- Push subscript through aggregate sum
- etc.
Accumulation

Suppose we have a couple of R statements:

\[
A \leftarrow C \ op \ D;
\]

\[
\ldots
\]

\[
\text{result} \leftarrow A \ op \ B;
\]
We can accumulate these statements:

\[
A \leftarrow C \text{ op } D;
\]

\[
\ldots
\]

\[
\text{result } \leftarrow A \text{ op } B;
\]

Does accumulating expressions expose new staging opportunities?
Preliminary results - staging

Compared staging to alternative approaches:
- typically outperforms “do everything at R”
- typically outperforms “do everything at SciDB”
- performance matched by greedy approach

<table>
<thead>
<tr>
<th>Initial object locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>
Preliminary results – expression rewriting and accumulation

Expression rewriting:
• Implemented associative and commutative rewrites
• Bonneville successfully identifying optimal plans using these general rewrite rules

Accumulation:
• Accumulated expressions are at least as good as individually-executed expressions
• Beneficial when execution location is arbitrary
More on accumulation:

Lowest staging cost: 700 data elements

Lowest staging cost: 700 to 750 data elements

Difference depends on where the root of subexpression (b) is staged
How does staging impact data movement?

- Generate and stage a wide variety of expressions, varying operators used, structure of expression
- Compare results with other methodologies, including greedy
- Measure runtime cost of staging

How does expression rewriting impact possible stagings?

- Measure runtime cost of expression rewriting
- Rewrite a range of expressions, measuring:
  - improvement over simple staging
  - runtime overhead

How does expression accumulation impact staging?

- Decompose a variety of expression types, varying “cut points”
- Stage decomposed expressions and compare to accumulated expressions
- Measure runtime cost of accumulation
- Determine “breakeven” point for accumulating expressions
Incorporating new physical properties into the cost model
  • Refinements of location
    - chunking scheme
    - linearization of chunked arrays
  • Refinements of size
    - storage format
    - compression

Tuning the optimizer and its rule set
  • What rules are doing the work?
  • What are interactions between:
    - rules?
    - optimizations?

Incorporating runtime considerations into the cost model
  • How long does operation x take at R?
  • How long does operation x take at SciDB?
Incorporating new physical properties into the cost model

- Refinements of location
  - chunking scheme
  - linearization of chunked arrays
- Refinements of size
  - storage format
  - compression

Tuning the optimizer and its rule set

- What rules are doing the work?
- What are interactions between:
  - rules?
  - optimizations?

Incorporating runtime considerations into the cost model

- How long does operation x take at R?
- How long does operation x take at SciDB?
Existing R integrations
- Ricardo
- RHIPE
- RIOT
- RICE

Query optimization
- System R and Exodus
- Volcano, Cascades, Columbia

Array systems
- Rewrite rules
- Storage features
Existing R integrations

• Ricardo
  • Integrates R and Hadoop/HDFS
  • Requires writing JAQL expressions

• RHIPE
  • Integrates R and Hadoop/HDFS
  • Requires refactoring R scripts into mapReduce paradigm

• RIOT
  • Integrates R and MySQL RDBMS

• RICE
  • Integrates R with SAP’s HANA RDBMS
  • R-Op
    • Requires writing calcModel programs
  • R-SHM
    • Reduces data movement
    • Orthogonal to our work
Thesis outline:

- **Introduction** – challenges of analyzing large datasets
- **Scalable Data Analysis Systems** – examining the three strategies, including existing systems
- **Arrays** – array models and theory
- **Optimization** – relevant issues in optimization, for both relational and array systems, including Bonneville
- **Agrios** – the Agrios platform, including subsections on R and SciDB
- **Experimental results** – Quantitative analysis of the effect and limits of staging, expression rewriting, and accumulation
- **Future work** – extensions of the work
- **Conclusion**
Thanks to David Maier, my committee, the DataLab, and the SciDB Team

Questions?