

SciDB Examples from Environmental Observation and Modeling

Bill Howe
Oregon Health & Science University
Center for Coastal Margin Observation and Prediction

November 19, 2008

Contents

1	Introduction	2
2	Interpolating Join	2
2.1	Other Formulations	5
3	Climatologies: Long-Term Monthly Tidal Averages	5
4	Timeseries Extraction from Unstructured Grids	7

1 Introduction

Datasets in environmental observation and modeling domain are fields defined over 4-D spacetime of the form $(x, y, z, t) \rightarrow (v_0, v_1, \dots, v_n)$, possibly extended with topology relationships. These datasets vary in which quantities are being measured or simulated (v_0, v_1, \dots, v_n) , but also in which dimensions, if any, are fixed and implicit. For example, a immobile buoy equipped with a Conductivity Temperature Depth (CTD) sensor at 1 meter below the surface has the form $(t) \rightarrow (salinity, temperature, elevation)$, since (x, y, z) need not be recorded. A profiling station can move vertically in the water column, but not horizontally, producing datasets of the form $(z, t) \rightarrow (salinity, temperature, elevation)$. Autonomous Underwater Vehicles (AUVs) are free to swim in any direction, so all attributes must be recorded. Simulations may involve 1-D, 2-D, or 3-D spatial domains.

The dependent variables (v_0, v_1, \dots, v_n) are also a source of heterogeneity. It is not unusual to see the same platform equipped with different sensors at different times during the year. The same variable may be recorded by multiple sensors for redundancy and calibration purposes (e.g., internal vs. external temperature.) A general yet efficient relational schema to store these data is non-obvious.

Each section below contains a separate example informed by common array operations used to manipulate environmental modeling datasets. These examples emphasize the query rather than the schema. In particular, the schema is usually simplified from 4-D to 1-D for clarity. Queries are expressed in an SQL-like syntax with array extensions, plus other syntaxes for discussion purposes. The discussion references the operators defined in the draft SciDB operator document [5].

2 Interpolating Join

When dimensions represent continuous domains (say, space or time), and attributes represent continuous functions over these domains, then a very common task is to interpolate the data from one array onto the dimensions of another array. The fundamental activity of an environmental modeler is to compare model results with observations; in this example, we consider simple 1-D timeseries.

There are three arrays in Figure 1: D represents observed data and has one dimension t and two attributes $time$ and $salt$, where $salt$ represents salinity. M represents model results and has an identical scheme as D . C represents the result of the interpolation operation. The $time$ attribute and $dsalt$ attribute are taken directly from D . The $msalt$ attribute holds the interpolated salinity from the model results. The NULL positions indicate that there is no unambiguous way to assign an interpolated modeled salinity to the given time.

Data	Model	Model onto Data																																																									
<table border="1" style="border-collapse: collapse; text-align: center;"> <thead> <tr><th></th><th><i>time</i></th><th><i>salt</i></th></tr> </thead> <tbody> <tr><td>D = 0</td><td>10</td><td>14.7</td></tr> <tr><td>1</td><td>20</td><td>15.2</td></tr> <tr><td><i>t</i> 2</td><td>30</td><td>14.9</td></tr> <tr><td>3</td><td>40</td><td>15.8</td></tr> <tr><td>4</td><td>50</td><td>16.5</td></tr> </tbody> </table> <p style="text-align: center;">$D(t) = (time, salt)$</p>		<i>time</i>	<i>salt</i>	D = 0	10	14.7	1	20	15.2	<i>t</i> 2	30	14.9	3	40	15.8	4	50	16.5	<table border="1" style="border-collapse: collapse; text-align: center;"> <thead> <tr><th></th><th><i>time</i></th><th><i>salt</i></th></tr> </thead> <tbody> <tr><td>M = 0</td><td>15</td><td>17.1</td></tr> <tr><td>1</td><td>25</td><td>17.8</td></tr> <tr><td><i>t</i> 2</td><td>35</td><td>17.6</td></tr> <tr><td>3</td><td>45</td><td>18.1</td></tr> </tbody> </table> <p style="text-align: center;">$M(t) = (time, salt)$</p>		<i>time</i>	<i>salt</i>	M = 0	15	17.1	1	25	17.8	<i>t</i> 2	35	17.6	3	45	18.1	<table border="1" style="border-collapse: collapse; text-align: center;"> <thead> <tr><th></th><th><i>time</i></th><th><i>dsalt</i></th><th><i>msalt</i></th></tr> </thead> <tbody> <tr><td>C = 0</td><td>10</td><td>14.7</td><td>NULL</td></tr> <tr><td>1</td><td>20</td><td>15.2</td><td>17.45</td></tr> <tr><td><i>t</i> 2</td><td>30</td><td>14.9</td><td>17.7</td></tr> <tr><td>3</td><td>40</td><td>15.8</td><td>17.85</td></tr> <tr><td>4</td><td>50</td><td>16.5</td><td>NULL</td></tr> </tbody> </table> <p style="text-align: center;">$C(t) = (time, dsalt, msalt)$</p>		<i>time</i>	<i>dsalt</i>	<i>msalt</i>	C = 0	10	14.7	NULL	1	20	15.2	17.45	<i>t</i> 2	30	14.9	17.7	3	40	15.8	17.85	4	50	16.5	NULL
	<i>time</i>	<i>salt</i>																																																									
D = 0	10	14.7																																																									
1	20	15.2																																																									
<i>t</i> 2	30	14.9																																																									
3	40	15.8																																																									
4	50	16.5																																																									
	<i>time</i>	<i>salt</i>																																																									
M = 0	15	17.1																																																									
1	25	17.8																																																									
<i>t</i> 2	35	17.6																																																									
3	45	18.1																																																									
	<i>time</i>	<i>dsalt</i>	<i>msalt</i>																																																								
C = 0	10	14.7	NULL																																																								
1	20	15.2	17.45																																																								
<i>t</i> 2	30	14.9	17.7																																																								
3	40	15.8	17.85																																																								
4	50	16.5	NULL																																																								

Figure 1: Two arrays of the same type. *M* stands for “model” and *D* stands for “data.” Each array has one integer dimension *t* and two attributes *time* and *salt*. The task is to interpolate the *salt* attribute of *M* at the *time* points of *D* to create *C*.

In SQL, this operation is rather awkward to express and difficult to optimize:

Listing 1: Interpolating join in SQL without dimension attributes

```

1 SELECT D.time , D.salt as dsalt ,
2       (hi.salt-lo.salt)*(D.time-lo.time)
3       / (hi.time-lo.time) + lo.salt as msalt
4 FROM D, M as lo , M as hi
5 WHERE lo.time <= D.time AND D.time < hi.time
6 AND NOT EXISTS (
7   SELECT * FROM M
8   WHERE (lo.time < M.time AND M.time < D.time)
9         OR (hi.time > M.time AND M.time > D.time)
10  )

```

This query requires two joins with *M*, and involves a correlated subquery which also scans *M*. If we assume the dimension *t* is available as an attribute, then we can avoid the correlated subquery in the WHERE clause by directly addressing the “next” tuple arithmetically:

Listing 2: Interpolating join in SQL with a contiguous integer dimension attribute

```

1 SELECT D.time , D.salt as dsalt ,
2       (hi.salt-lo.salt)*(D.time-lo.time)
3       / (hi.time-lo.time) + lo.salt as msalt
4 FROM D, M as lo , M as hi
5 WHERE D.time >= lo.time AND D.time < hi.time
6       AND lo.t + 1 = hi.t

```

To express this operation in SciDB, these SQL formulations can be translated directly into relational-style algebraic expressions using Join and Apply. This translation assumes that index arithmetic is allowed in the Join predicate (e.g., $lo.t = hi.t + 1$). Although there does not appear to be an explicit example of this syntax, there does not seem to be anything preventing it. Although a join-based formulation is possible, it is not the most natural way to express the algorithm for users familiar with arrays. Further, handling arrays as sets is not likely to lead to an efficient implementation.

In a general purpose programming language equipped with arrays, the operation is arguably simpler, more familiar to scientists, and, on very small arrays that fit in memory, much faster:

```

1  for i in |D|:
2    for j in |M| - 1:
3      if M[j] <= D[i].time < M[j+1].time:
4          D[i].msalt = (M[j+1].salt - M[j].salt) \
5                      * (D[i].time - M[j].time) \
6                      / (M[j+1].time - M[j].time) \
7                      + M[j].salt

```

However, this method is highly imperative and therefore difficult to optimize and parallelize when M and D are very large and distributed across a cluster of SciDB nodes. (This 1-D example admits a fairly simple parallelization strategy, but the general case is difficult to analyze.)

Using the SciDB operators, it is not obvious how to implement this style of algorithm. For example, the Apply operator is described as an element-wise operation only, which precludes the index arithmetic style of programming. Index arithmetic makes a variety of algorithms over “neighborhoods” of elements very easy to express. These algorithms should probably be considered an important requirement for SciDB, and should therefore probably be naturally supported in the interface in the interest of making “easy things easy.”

Setting aside the SciDB operators, consider an SQL-like syntax that directly supports array-style operations. Instead of traditional tuple aliases that iterate over all tuples, we assume that index aliases are used to iterate over the dimensions of each array. For example,

```

1  SELECT D[i].time ,
2         D[i].salt as dsalt ,
3         (m[j+1].salt - m[j].salt) * (D.time - m[j].time)
4         / (m[j+1].time - m[j].time) + m[j].salt as msalt
5  FROM D[t as i], M[t as j] as m — arr and dim aliases
6  WHERE D[i].time >= m[j].time AND D[i].time < m[j+1].time

```

2.1 Other Formulations

Salem and Marathe proposed three operators for their Array Manipulation Language [4]: The interpolation operation can be expressed using a composition of the MERGE and APPLY operators, but the expression is very complicated and is therefore omitted.

In RasQL, the query language used in the RasDaMan system [1], array elements usually hold only primitive values, so each attribute is separated into its own array. The `marray` constructor builds an array.

```

1 SELECT marray i in sdom(Dtime), j in sdom(Mtime)
2     values Msalt[j]
3           + (Msalt[j+1] - Msalt[j])
4           * (Dtime[i] - Mtime[j])
5           / (Mtime[j+1] - Mtime[j])
6 FROM Dtime, Dsalt, Mtime, Msalt
7 WHERE Mtime[j] <= Dtime[i] < Mtime[j+1]
```

Libkin and Wong's Array Query Language [3] uses a comprehension syntax. Arbitrary nesting is also possible, so we assume each array element is tuple.

$$[\text{interpolate}(D[i], M[j], M[j+1]) \mid i < \text{len}(D), j < \text{len}(M) - 1]$$

where

$$\text{interpolate}(X, lo, hi) =$$

$$lo.\text{salt} + \frac{(hi.\text{salt} - lo.\text{salt})(X.\text{time} - lo.\text{time})}{(hi.\text{time} - lo.\text{time})}$$

3 Climatologies: Long-Term Monthly Tidal Averages

Figure 2 shows the plume of fresh water outside the mouth of the Columbia River Estuary for two different months averaged over 10 years. Only the low tide information is used to compute the average. The complete problem involves a 4-D array, but we can restrict the domain to a single node without losing the salient features. Restricted to a single node, the problem is again 1-D. The task is to compute the water elevation at each low tide event, then average by month. We estimate the elevation at low tide by looking at the region where the first derivative crosses zero Figure 3. To estimate the first derivative, we use a finite differences method, which just means that we use a neighborhood around each element to estimate the slope. We assume an array D of the form in Figure 1, but with an attribute *elev* in place of *salt*. In the hypothetical Array SQL pseudocode:

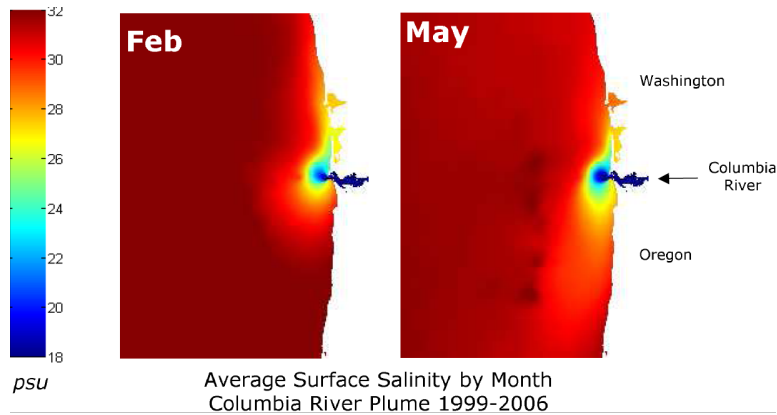


Figure 2: Average surface salinities at low tide for February and May averaged over 7 years. The 4-D dataset occupies 7TB.

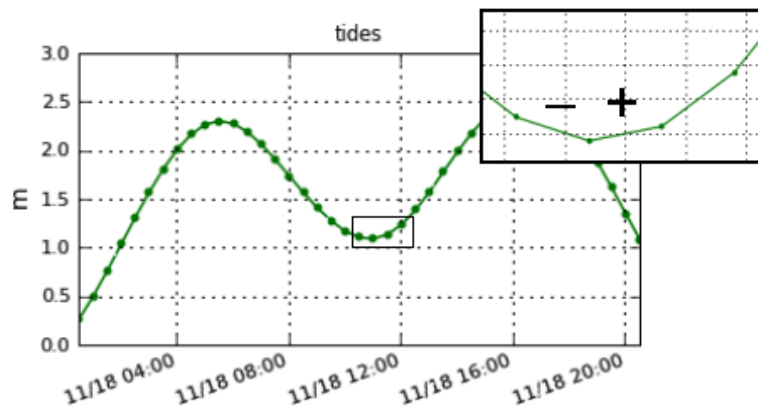


Figure 3: Tidal signal near the Columbia River Estuary. Low tide elevation can be estimated using a query that identifies the array elements at which the first derivative changes from negative to positive (inset).

Listing 3: Estimating the first derivative of the tidal signal in Figure 3.

```

1 SELECT D[ i ]. time ,
2        D[ i ]. elev ,
3        (D[ i ]. elev - D[ i - 1 ]. elev )
4        / (D[ i ]. time - D[ i - 1 ]. time ) as dedt
5 FROM D[ t as i ]
6 WHERE month( D[ i ]. time ) IN ( ' February ' , ' Mach ' )

```

To compute the low tide elevation, we use interpolation as in Section 2, but here we just use the value at which the derivative changes direction. Assuming the result of Listing 3 is stored in an array E , we can extract the low tide values with a simple

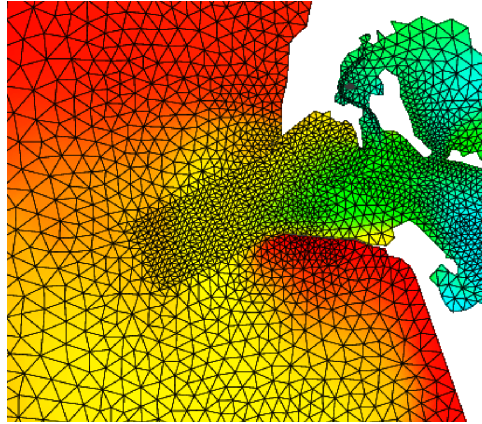


Figure 4: Environmental modelers frequently use unstructured grids that are superior at representing complex, multiscale domains. However, these grids complicate processing, since the topology relationships must be represented explicitly.

WHERE clause, and the monthly average can be expressed with a GROUP BY.

```

1 SELECT month(E[ i ]. time ), E[ i ]. elevation
2     FROM E[ t as i ]
3  WHERE E[ i ]. dedt <= 0 < E[ i +1 ]. dedt
4  GROUP BY month(E[ i ]. time )

```

4 Timeseries Extraction from Unstructured Grids

In Section 2, we joined model results and observed data. However, the model results are not natively available as a 1-D timeseries at the exact same (x, y, z) location as the physical sensor. This timeseries must be extracted from a large 3-D field discretized into a collection of polygons called an *unstructured grid*. Three different grid types used in environmental modeling are shown in Figure 5.

Ignoring depth and time, we have a 2-D horizontal grid representing the water's surface as in Figure 4. The task is to locate the triangle containing a given point, then interpolate to derive its associated value. The 3-D grid is not rectilinear, however, and cannot be completely represented by a single array. Instead, we use two arrays: one for the nodes of the grid, and one for the incidence relationship between cells of different dimensions. Figure 6 illustrates a simple example of this representation. A survey of data structures for grids can be found [2].

The two arrays are $Nodes(id) \rightarrow (x, y, bathymetry)$ and $Cell(id) \rightarrow (n1, n2, n3)$. The attributes (x, y) give the location of the node, and *bathymetry* indicates the mean depth of the water at that location. The strategy is to join the node data to

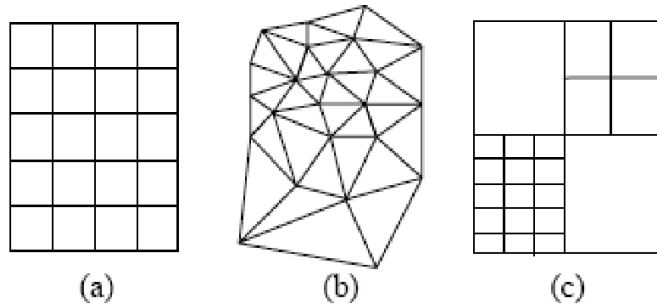


Figure 1.7: (a) A structured grid. (b) An unstructured grid. (c) A hierarchical grid.

Figure 5: Three types of grid used in environmental modeling. (a) A structured grid that can naturally be represented by a single array. (b) An unstructured grid requiring explicit topology relationships. (c) A multiresolution grid.

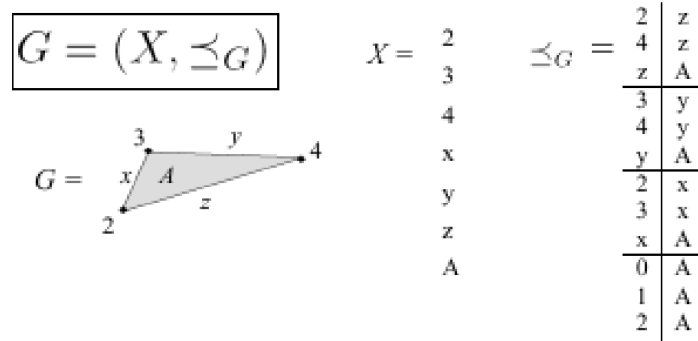


Figure 6: Representing an unstructured grid with two arrays. X holds the node ids, and \preceq_X holds the incidence relationship between cells.

the triangle cells, then just use a magic contains operator to test for containment. The ids of the nodes are stored as values in the cells array as an implicit FK. This operation does not exercise the array features of the language.

```

1 SELECT interpolate($x, $y, triangle(N1[i], N2[j], N3[k]))
2   FROM Cell[c] as C,
3       Node[id as i] as N1,
4       Node[id as j] as N2,
5       Node[id as k] as N3
6 WHERE C[c].n1 = i AND C[c].n2 = j AND C[c].n3 = k
7       AND contains($x, $y, triangle(N1[i], N2[j], N3[k]))

```

References

- [1] P. Baumann. A database array algebra for spatio-temporal data and beyond. In R. Y. Pinter and S. Tsur, editors, *NGITS '99: Proceedings of the 4th Workshop on Next Generation Information Technologies and Systems*, pages 76–93, Zikhron-Yaakov, Israel, July 1999.
- [2] B. Howe. *GridFields: Model-Driven Data Transformation in the Physical Sciences*. PhD thesis, Department of Computer Science, Portland State University, Portland OR, USA, 2007.
- [3] L. Libkin, R. Machlin, and L. Wong. A Query Language for Multidimensional Arrays: Design, Implementation, and Optimization Techniques. In *Proceedings of ACM SIGMOD*, pages 228–239, 1996.
- [4] A. P. Marathe and K. Salem. A language for manipulating arrays. In *Proceedings of VLDB*, pages 46–55, Athens, Greece, August 1997. Morgan Kaufmann.
- [5] S. Zdonik, D. Maier, J. Blakeley, and A. Szalay. Standard Operators in SciDB (early draft), 2008. personal communication.