ADR Customization Interface

Joel Saltz

Alan Sussman

Tahsin Kurc

University of Maryland, College Park

and

Johns Hopkins Medical Institutions

http://www.cs.umd.edu/projects/adr

Data Loading Service

- **Provides support for loading datasets into ADR**
  - A set of utility programs to load and register the datasets
- **Loading data into ADR**
  - partition the dataset into data chunks,
  - compute placement information,
  - create an ADR index,
  - move data chunks to the disks according to the placement.

Loading Datasets

- **Raw dataset**
  - No partitioning into chunks
  - No placement information and no index
- **Half-cooked dataset**
  - Already partitioned into chunks
  - No placement information and no index
- **Fully-cooked dataset**
  - Already partitioned into chunks
  - User-defined placement information, chunks already declustered
  - User-defined index

Data Loading Service

- **User must partition the dataset into chunks**
- **For a fully cooked dataset, User**
  - moves the data and index files to disks (via ftp, for example)
  - registers the dataset using ADR utility programs
- **For a half cooked dataset, ADR**
  - computes placement information using a Hilbert curve-based declustering algorithm,
  - builds an R-tree index,
  - moves the data chunks to the disks
  - registers the dataset

Data Loading Service

- **A master manager**
  - computes the placement information
  - manages movers
- **Movers**
  - run on each back-end node
  - responsible for copying the data chunks to local disks
  - builds R-tree index for data chunks on local disks
  - currently, each mover should be able to access all the data files in a half-cooked dataset (e.g., over a shared file system)

Loading Half-Cooked Datasets

- **User has to provide:**
  - Name files -- ASCII file
    - lists the names of the files that contain the data chunks
  - Linear index files -- ASCII
    - contains unordered list of m

Alan Sussman:

what else beside mbrs here? - ALS

  - Loader command file -- ASCII file
    - contains a list of all half-cooked datasets to be loaded
  - Mover configuration file -- ASCII file
    - lists configuration information for all movers
      - number of disks accessible by a mover
      - directories to write files on local disks, etc.

Name Files

```
Path/data-file1          #  name of the data file 1
Path/data-file2          #  name of the data file 2
.
.
.
Path/data-fileN          # name of the data file N
```

Linear Index Files

```
FORMAT  NDIMS  NENTRIES          # header
# record for entry 1
low[0] low[1] … low[ndims-1]     # MBR for entry 1
high[0] high[1] … high[ndims-1]
nblocks                          # number of physical blocks
file_id offset size              # info for physical block 1
file_id offset size              # info for physical block 2
…
nbytes                           # size of user defined data
userdata                         # user data
# record for entry 2
…
```

Loader Command File

```
# dataset record for entry 1
dataset_name          # name of the dataset
dataset_prefix        # prefix for data files in ADR
index_name            # name of the index
index_prefix          # prefix for index files in ADR
num_metadata          # number of <name, linear index> files
name_file1            # name file 1
index_file1           # linear index file 1
…
name_fileN            # name file N
index_fileN           # linear index file N
```

Mover Configuration File

```
# record for mover 1
mover_id num_disks     #  <node id, number of local disks>
path1                  # directory to store files on disk 1
path2                  # directory to store files on disk 2
…
pathN                  # directory to store files on disk N
dataset-catalog-prefix # prefix for dataset catalog file
index-catalog-prefix   # prefix for index catalog file
dataset-config-set     # dataset configuration set file name
dataset-config         # dataset configuration file
```

Mover Configuration File

- Each data configuration file lists the files local to a back-end node
- Data configuration set file lists the names of the data configuration files.
- Dataset catalog file contains a list of data files for all datasets loaded/registered in ADR
- Index catalog file contains a list of all index files loaded/registered in ADR
- ADR back-end uses these files at runtime to get information about datasets/indexes in ADR

Registering A Dataset

- **Description of a dataset in ADR consists of**
  - dataset id: given by ADR, unique, used in ADR queries
  - dataset name: given by the user
  - dataset description: a short description of the dataset
  - iterator name: the iterators to access data elements
  - index name: the name given by the user for the index
  - index id: given by ADR, unique, used in ADR queries
- **ADR provides utility programs to register datasets**

ADR Front-end

- **Interacts with application clients and the ADR back-end**
  - Receives requests from clients and submit queries to ADR back-end
- **Provides services for clients**
  - to connect to ADR front-end
  - to query ADR for information on datasets and user-defined methods in ADR
  - to create and submit ADR queries

Connecting to ADR front-end

```
#include <t2_frontend.h>

class T2_FrontEnd {
        bool connectT2FrontEndbyHostname(…)
        bool connectT2FrontEndbyAddress(…)
        void disconnectT2FrontEnd(…)

        T2_FrontEndError getErrorVal(…)
        char *errorValToString(…)

        u_int getNumberBackEndNodes(…)
}
```

```
#include "t2_frontend.h"

T2_FrontEnd fe;
fe.connectT2FrontEndByHostname(hostname, port); // connect to ADR frontend
...
 // inquire of ADR front-end about functions
const char svm_keyword[] = "t2-svm-example";
svm_inquire_functions(fe, svm_keyword, pid, accid, aggid);
...
// inquire of ADR front-end about datasets (images)
svm_inquire_datasets(fe.svm_keyword, pid, accid, aggid, thumbnail_dir, images);
...
// interact with an SVM client, and when need to create an ADR query object
// for image i, resolution z, and a query region starting from (x,y) of w
// pixels wide and h pixels high ...
GenerateQuery(fe, packno, packtype, x, y, w, h, z, hostname, backendport, images[i]);
…
fe.disconnectT2FrontEnd();                    // disconnect from ADR front-end
```

Querying ADR for Datasets and User-defined Methods

```
class T2_FrontEnd {
        // Inquiry for dataset information
        inquireDatasetExactMatch(…)
        inquireDatasetRegExp(…)

        // Inquiry for user-defined functions
        inquireFunctionExactMatch(…)
        inquireFunctionRegExp(…)
}
```

Querying for Datasets

- **The result of a query for dataset meta-data contains**
  - dataset id: used in ADR query
  - dataset name: name given by the user
  - dataset description: short description of dataset
  - iterator name: list of the names of the iterators
  - index name: index name given by the user
  - index id: used in ADR query
  - dataset blob: user-defined binary object, e.g., a thumbnail image
  - dataset blob size: size of binary object

Querying for User-defined Functions

- **The inquiry methods take a user-defined name or a regular expression and a “function type”**
- **Valid function type parameters**
  - T2_UDF_Unknown: all functions
  - T2_UDF_AccMeta: accumulator meta-data object
  - T2_UDF_Aggregation: aggregation functions
  - T2_UDF_Projection: projection functions

Querying for User-defined Functions

- **The result of an inquiry for a function consists of**
  - function id: given by ADR, used in the ADR query
  - function name: given by the user
  - function description: a short description of the user-defined function

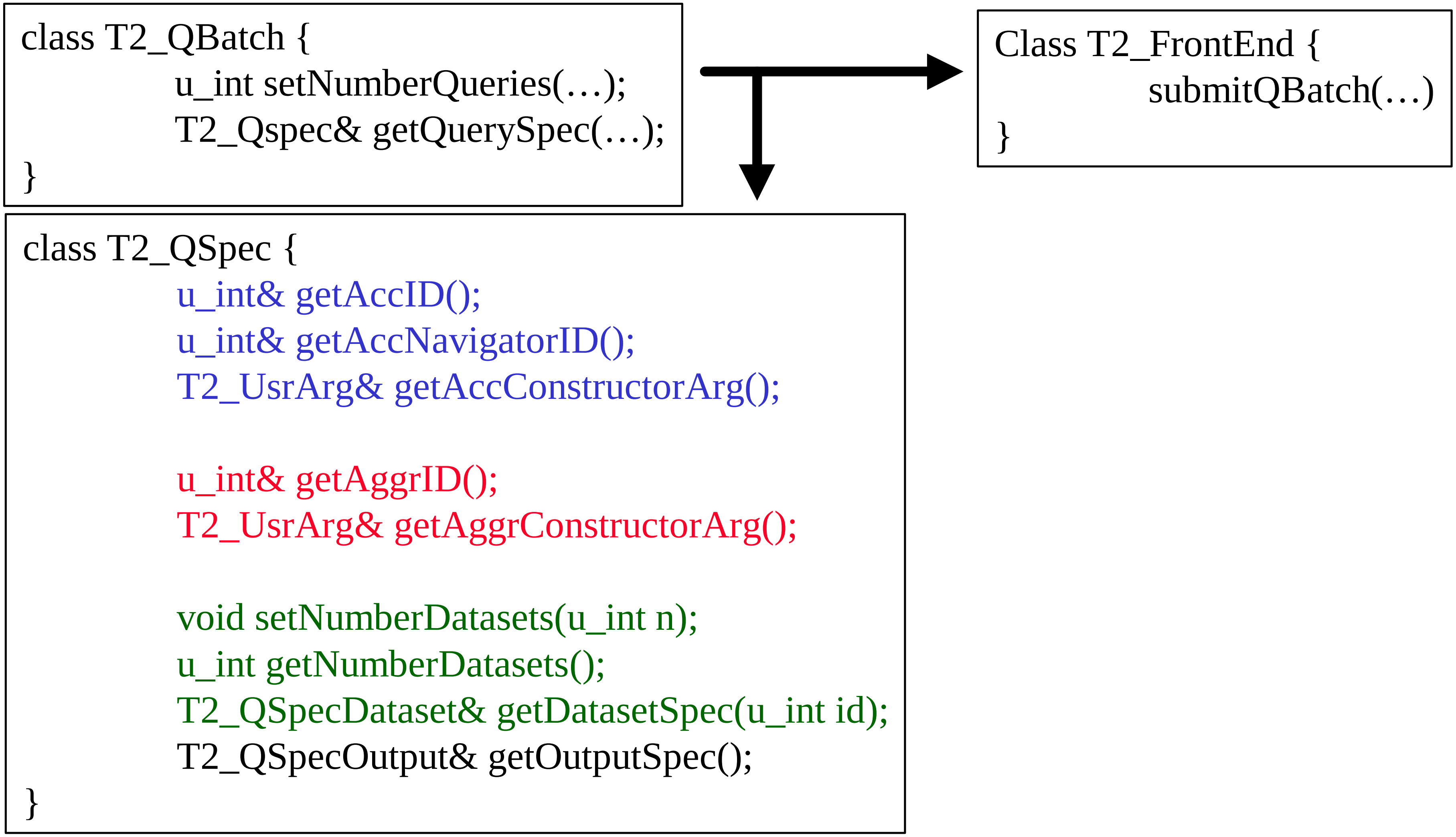
```
svm_inquire_functions(T2_FrontEnd& fe, const char* keyword, u_int& pid,
                                   u_int& accid, u_int& aggid)
{
  T2_FEFunctionInquiryResults func_results;        // to hold results
  const int func_fields = T2_FEInquiry::function_id_field |
                                   T2_FEInquiry::function_name_field;
  fe.inquireFunctionRegExp(keyword, T2_UDF_Unknown, func_fields, func_results);

  for (u_int i=0; i<func_results.getNumberEntries(); i++) {
    T2_FEFunctionEntry& entry = func_results[i];
    switch (entry.getFunctionType()) {
      case T2_UDF_AccMeta:           // accumulator meta object
                  if (accid == 0) accid = entry.getFunctionID();
                  break;
      case T2_UDF_Projection:        // projection function
                  if (pid == 0)   pid = entry.getFunctionID();
                  break;
      case T2_UDF_Aggregation:       // aggregation function
                  if (aggid == 0) aggid = entry.getFunctionID();
                  break;
}}}
```

ADR Query

- **An ADR query consists of**
  - a reference to an accumulator meta-data object/function
  - a reference to an aggregation function
  - references to one or more input datasets
    - a reference to a dataset iterator function
    - a reference to index
    - a reference to a projection function
    - a range query, a multi-dimensional bounding box, defined in the underlying multi-dimensional attribute space of the input dataset
  - user-defined parameters for the functions
  - specification for how to handle the output (e.g., send the output to the client via sockets)

ADR Query



```
class T2_QBatch {
        u_int setNumberQueries(…);
        T2_Qspec& getQuerySpec(…);
}
```

```
Class T2_FrontEnd {
        submitQBatch(…)
}
```

```
class T2_QSpec {
        u_int& getAccID();
        u_int& getAccNavigatorID();
        T2_UsrArg& getAccConstructorArg();

        u_int& getAggrID();
        T2_UsrArg& getAggrConstructorArg();

        void setNumberDatasets(u_int n);
        u_int getNumberDatasets();
        T2_QSpecDataset& getDatasetSpec(u_int id);
        T2_QSpecOutput& getOutputSpec();
}
```

```
GenerateQuery(…)
{
  T2_QBatch qbatch(1);                       // only one query per batch
  T2_QSpec& qspec = qbatch.getQuerySpec(0);
  qspec.setNumberDatasets(1);            // only access one dataset
…
  qspec.getAggrID() = ie->getAggregationID();
  qspec.getAggrConstructorArg().allocBuffer(sizeof(u_int)*2);
  write_arg.open(qspec.getAggrConstructorArg());
  write_arg << packno << packtype;
  write_arg.close();
...
 T2_Box& inbox = dataset_spec.getQueryBox();  // set the input query box
  inbox.setNumberDimensions(2);
  inbox.getLow()[0] = x;                 inbox.getLow()[1] = y;
  inbox.getHigh()[0] = x+w;          inbox.getHigh()[1] = y+h;
…
  T2_QSpecOutput& outspec = qspec.getOutputSpec();
  outspec.setOutputHandleType(t2_oSocket);
  outspec.setHostName(hostname);         // store the hostname of the client
  outspec.getPortNumber() = backendport; // store the port number client listens to
  outspec.useBigEndianOutput();          // use big endian for output for Java client
  outspec.disableT2Protocol();          // do not use ADR standard protocol
}
```

Customizing ADR Back-end

- **User has to provide implementations for**
  - Dataset service
  - Indexing service
  - Attribute space service
  - Data aggregation service
- **Implementation of ADR services is based on C++ class inheritance, and virtual functions**
- **Constructor functions to create user-derived objects**

Dataset Service

- **Manages user-defined dataset iterators**
- **An iterator is used to iterate through data elements in a data chunk**
- **A user-defined iterator should**
  - understand the structure of the data chunk
  - output the data values and coordinates of each data element in the data chunk

Dataset Service

```
#include <t2_dataset.h>
class T2_Dataset {
        // generate iterator
        genIterator(…);
}
class T2_Iterator {
        // get next element
        getNextElement(…);
}

T2_DatasetConstructor userDatase(…);
```

Indexing Service

- **Manages the default and user-defined indexing methods**
- **An index is used to efficiently locate the data chunks that intersect a range query**
- **Index files contain**
  - minimum bounding rectangle of each chunk, and
  - user-defined meta-data for each chunk (optional)

Indexing Service

```
#include <t2_index.h>
class T2_Index {
        // initialize search
        fetchInit(…);
        // get the next element
        fetch(…);
}

T2_IndexConstructor userIndex(…);
```

```
#include "t2_index.h"

class svm_ImageIndex: public T2_Index {
  T2_Array<FILE*> fps;
  u_int curfp;
};

T2_UDFRet  svm_ImageIndexConstructor(...)
{  // ignore system, arg
    idxp = (T2_Index *) new svm_ImageIndex(fd);
    return T2_UDFRet_OK;
}

svm_ImageIndex::svm_ImageIndex(T2_Array<int> fds)
{
  // turn all the file descriptors into file streams and store them in fps[]
  for (u_int i=0; i<fds.getNumberElements(); i++)
     fps[i] = fdopen(fds[i], "r");
}

T2_UDFRet svm_ImageIndex::fetchInit(...)
{
  curfp = 0;                // reset counter to start from first index file
  fseek(fps[curfp], 0, SEEK_SET);       // rewind the file
  return T2_UDFRet_OK;
}
```

```
svm_ImageIndex::fetch(…)  {
  mbr.setNumberDimensions(2);    // each input is 2-dim
  while (curfp < fps.getNumberElements()) {
    while (!fps[curfp].eof()) {
      // read from fps[curfp] all the stored information about this cluster
      my_read(fps[curfp], x_pos, y_pos, width, height, pixelsize, offset, size, fid);
      // compute mbr of the cluster
      mbr.getLow()[0] = x_pos;   mbr.getLow()[1] = y_pos;
      mbr.getHigh()[0] = x_pos + width - 1; mbr.getHigh()[1] = y_pos + height - 1;

      if (qr ^ mbr) {           // cluster mbr intersects with query region
         // store x_pos and y_pos in info, just for demonstration purpose
         info << x_pos << y_pos;
         // set one block request for this cluster
         chk << T2_BlockRequest(fid, offset, size);

         eod = false;           // set end-of-data to false
         return T2_UDFRet_OK;
       }}          // while my_read()

   eod = true;   // no more data
   return T2_UDFRet_OK;
}
```

Attribute Space Service

- **Manages user-defined projection functions**

- **A projection function projects a point in the input space to a set of points in the output space.**

Attribute Space Service

```
#include <t2_prj.h>
class T2_ProjectFuncObj {
        // project a point to a set of points
        project(…);
        // get information about attribute spaces
        getNumberInputDimensions();
        getNumberOutputDimensions();
        // project a box in input space to a box in output space
        projectBox(…);
}

T2_ProjectFuncConstructor userProject(…);
```

```
#include "t2_prj.h"

class svm_ImageProjection: public T2_ProjectFuncObj {
 u_int x_orig, y_orig, x_size, y_size;
};

T2_UDFRet svm_ImgPrjConstructor(...)
{
  arg >> xo >> yo >> xs >> ys;
  prjfunc = (T2_ProjectFuncObj *) new svm_ImageProjection(xo, yo, xs, ys);
  return T2_UDFRet_OK;
}
```

```
T2_UDFRet
svm_ImageProjection::project(const T2_System& system,
                             const T2_Box& iqr, const T2_Region& tilereg,
                             const T2_Point& input_pt,
                             bool& succeed, T2_Region& output_rg)
{
  succeed = false;
  if (!iqr.contains(input_pt))          // point not inside query window
     return T2_UDFRet_OK;

  T2_Point output_pt(2);                // a point by projecting input_pt
  output_pt[0] = (input_pt[0] - x_orig) / x_size;
  output_pt[1] = (input_pt[1] - y_orig) / y_size;
  if (tilereg.contains(output_pt) == true) {
     output_rg.setNumberDimensions(2);
     output_rg.growMaxSetSize(1);       // grow region set to store one point
     output_rg << output_pt;
     succeed = true;
 }
  return T2_UDFRet_OK;
}
```

Data Aggregation Service

- **Provides interface**
  - to create and manipulate user-defined accumulator objects,
  - to implement aggregation operations,
  - to create output data structures,
  - to implement functions to convert accumulator values to the final output values.

Data Aggregation Service -- Accumulator

- **An accumulator is a user-defined data structure to hold intermediate results.**
- **Each accumulator is associated with**
  - Accumulator meta-data object
    - stores the meta-data for the accumulator, e.g., minimum bounding rectangle of the accumulator
    - provides methods to partition the accumulator into tiles
  - Accumulator object
    - encapsulates the accumulator data structure
    - provides methods to create iterators to access individual elements
  - Accumulator Iterator
    - used to access individual accumulator elements.

Data Aggregation Service -- Accumulator

```
#include <t2_acc.h>
class T2_AccMetaObj {
            stripMine(…); // partitioning accumulator into tiles
            allocAcc(…); // allocating an accumulator tile
}
class T2_Accumulator {
            navigateAll(…);  // create iterator to access all elements
            navigate(…); // create iterator to access elements in a region
}
class T2_AccIterator {
            getNextElement(…); // get the next accumulator element
}
T2_AccMetaConstructor userAccMeta(…);
```

```
#include "t2_acc.h"
class svm_ImageAcc: public T2_Accumulator {
  u_int x_res, y_res;      // width and height of entire output image
  u_int x_pos, y_pos,     // coordinates of the starting pixel in the
                           // output space
        width, height,     // width and height of this accumulator
        pixelsize;         // number of bytes per pixel
};

class svm_ImageAccIterator: public T2_AccIterator {
  svm_ImagePixel pixel;
};

class svm_ImageAccMeta: public T2_AccMetaObj {
  u_int width, height,        // size of the entire output image (in # pixels)
        pixelsize;             // number of bytes per pixel
};
```

```
T2_UDFRet svm_ImageAccMeta::stripMine(… size_t mem, …) {
  mem = mem - sizeof(svm_ImageAcc);      // reserve space for an svm_ImageAcc obj

  // for simplicity, we are just showing the code that partition the image by rows
  size_t rowsize = width * pixelsize;
  u_int y_mem = mem / rowsize;          // # rows that fit inside memory
  u_int ntiles = height / y_mem;

  for (u_int i=0, y_start=0; i<ntiles; i++) \{
    T2_Box mbr(2); T2_Point& low = mbr.getLow(); T2_Point& high = mbr.getHigh();
    mbr.getLow()[0] = 0;
    mbr.getLow()[1] = (T2_ShapeCoord_t) y_start;
    mbr.getHigh()[0] = (T2_ShapeCoord_t) (width - 1);

    u_int y_end = y_start + y_mem - 1;
    if (y_end >= height)
      mbr.getHigh()[1] = (T2_ShapeCoord_t) (height - 1);
    else
      mbr.getHigh()[1] = (T2_ShapeCoord_t) y_end;

    tile_mbrs << mbr;
    y_start = y_end + 1;
  }
  return T2_UDFRet_OK;
}
```

Data Aggregation Service -- Aggregation

- **Methods to aggregate input elements and accumulator elements --** ***Local Reduction Phase***

- **Methods to aggregate accumulator elements --** ***Global Reduction Phase***

- **Convert accumulator values to the final output values --** ***Output Handling Phase***

Local Reduction Functions

```
#include <t2_aggr.h>
class T2_AggregateFuncObj {
        // Initialize accumulator elements
        aifElem();
        aifAcc();
        // aggregate input elements with accumulator elements
        dafElem();
        dafAcc();
}

T2_AggregateFuncConstructor userAggregateFunc(…);
```

```
#include "t2_aggr.h"
class svm_aggregation: public T2_AggregateFuncObj {
  u_char *my_bytes;                 // the assigned bytes to combine on local proc
  size_t my_nbytes;                 // the number of bytes assigned to local proc
  u_int my_x_pos, my_y_pos,         // the position of the 1st assigned pixel
                                    // in the entire output image
        my_width,                   // width of local assigned sub-image
        my_height;                  // height of local assigned sub-image

};

T2_UDFRet svm_aggregation(...) {
  u_int packet_no, packet_type;
  arg >> packet_no >> packet_type;
  afp = (T2_AggregateFuncObj *) new svm_aggrMax(packet_no, packet_type);
  return T2_UDFRet_OK;
}
```

```
T2_UDFRet svm_aggregation::aifElem(const T2_QueryInfo& qinfo, void* accElem) {
  t2_Assert (accElem != NULL);
  svm_ImagePixel& pixel = *((svm_ImagePixel *) accElem);
  for (u_int i=0; i<pixel.getNumberBytes(); i++)
     pixel[i] = 0;

  return T2_UDFRet_OK;
}

T2_UDFRet svm_aggregation::dafElem(…) {
  const svm_ImagePixel& input_pixel = *((const svm_ImagePixel *) dataElem);
  svm_ImagePixel& output_pixel = *((svm_ImagePixel *) accElem);
  for (u_int i=0; i<input_pixel.getNumberBytes(); i++)
      if (input_pixel[i] > output_pixel[i])
         output_pixel[i] = input_pixel[i];
  return T2_UDFRet_OK;
}
```

Global Combine Functions

```
#include <t2_aggr.h>
class T2_AggregateFuncObj {
        // is global combine phase needed?
        needGlobalCombine();
        // pack a subset of local accumulator elements
        // into buffers for other processors
        fillAccMsgBuffer(…);
        // unpack and process accumulator values
        // received from other processors
        processAccMsg(…);
}
```

Output Handling

```
#include <t2_aggr.h>
#include <t2_output.h>
class T2_AggregateFuncObj {
        // create output object, convert accumulator values
        // to output values
        finalize(…, T2_Output* output_obj);
}

class T2_Output {
        // size of the output buffer required to pack output data
        getOutputBufferSize();
        // pack output data into contiguous buffers to be sent to
        // the client or written to the disks.
        flushOutput(…);
}
```

```
#include "t2_output.h"
class svm_OutputSubImage: public T2_Output {
  u_int packet_no,                    // packet number from aggregation func
        packet_type;                  // packet type from aggregation func

  u_int nprocs,                       // # back-end nodes
        ntiles,                       // # output tiles
        x_pos, y_pos,                 // position of the sub-image in the
                                      // entire output image
        sub_x_res, sub_y_res;         // width and height of sub-image
  u_char* bytes;                      // pointer to bytes of the sub-image
                                      // pixels (the bytes are owned by
                                      // another object, so no need to
                                      // delete them here)
  u_int pixelsize,                    // number of bytes per pixel
        maxval;                       // max value for each component byte
  u_int x_res, y_res;                 // width and height of entire output
};
```

```
size_t svm_OutputSubImage::getOutputBufferSize(...) {
  // need to allocate buffer for the header
  return sizeof(packet_no) + sizeof(packet_type)
         + sizeof(nprocs) + sizeof(ntiles) + sizeof(x_res) + sizeof(y_res)
         + sizeof(x_pos) + sizeof(y_pos) + sizeof(sub_x_res)
         + sizeof(sub_y_res);
}

T2_UDFRet svm_OutputSubImage::flushOutput(...) {
  const char *p1 = buf.getCurrentPosition(),    // pointer to header
  const char *p2;                               // a temp pointer
  buf << nprocs << ntiles << x_res << y_res << x_pos << y_pos
        << sub_x_res << sub_y_res;
  p2 = buf.getCurrentPosition();
  ptr.insertDataPointer(p1, p2 - p1);           // insert pointer to header

  u_int npixels = sub_x_res*sub_y_res;
  if (npixels > 0)
    ptr.insertDataPointer((const char*) bytes, npixels * pixelsize);
  return T2_UDFRet_OK;
}
```