1 High Level Design

1.1 Layers

The Clusterer is organized in three (very) separate layers, each layer has some clear responsability and communication between layers is strictly limited (i.e. there is exactly one point of communication in each layer that can communicate either with the next layer up or the previous one down). I refer you to the class diagram provided at `/trunk/doc/`.

The three layers, and their responsabilities, are:

1. **Main Application** : This layer contains all the application specific classes, this includes concrete instances of the abstract Distributable Jobs (which encapsulates a non atomic work) as well as all data specific classes (including Data description and handling). It is important to note that only code written for this layer would eventually have to be changed if one were to shift the problem domain to something completely different, but wanted to mantain the underlying scheduling policy and client management scheme.

2. **Work Distribution Management** : All Distributable Jobs will eventually separate themselves in a number a Job Units (an atomic work unit). It is the responsability of this layer to handle the assignment of these units to a specific processing client (just Client from now on) and later inform the results thus obtained. From this layer there is no relevance as to where Clients come from or are located (and can be therefore implemented using a variety of multiprocessing paradigms such as processes in a single core, threads of some sort, BOINC clients, MPI, etc.). Also, the type of problem to be solved is irrelevant from this layer's point of view. This layer would be subsequently modified if one were to modify the distribution policies of Job Units. At this point, you have the abstract definition of a *DistributableJob* and the *Distributer* itself, which will work as the JobUnit scheduler of the system.

3. **Client Management** : Responsabilities of the layer include handling client registration and status throughout the application's runtime. The layer does not care about specific problem details or the Job Units themselvs, it will only function as a bridge between single Clients and the Job Units sent from layer 2 classes. Code in this layer can thus be modified if one wishes to change the implementation of how specific Clients are handled or where they exist. Since in this specific application Clients happen to be based on the volunteer computing model of BOINC, you can find interfaces that communicate with a BOINC server at this stage.

From this abstract view of the design, it should be noted that layer 3 alone constitutes a particular client management scheme. It could be implemented using threads or processes in a single core, volunteer computing through internet using BOINC or even a MapReduce distributed system like Hadoop. These separate implementations of layer 3 must be interchangeable, one must be able to swap them and the problem must still be solved correctly.

Layer 2 could implement different scheduling policies for assigning specific Job Units, generally the implementer should know that many details of the layer should be tuned to work with the specifics of the application (or layer 1). For example we would want a large queue for job units if we knew beforehand that the application will be generating a great deal of small job units. The implementer should also tune constants describing the size of the generated job units to balance the tradeoff between processing time and communication time (in the case that communication is an issue to be considered, as is with BOINC). So, Job Units shouldn't be so big that it takes a lot of time to go from the Distributer to a particular client or so small that it can be computed instantly.

It should be noted that in the entire system, the total time of computation of a single Job Unit in an ideal setting would be:

$$T_{job-unit} = T_{send} + T_{compute} + T_{recieve} + T_{handle-results}$$

The first and third being communication dependant, the second one being client dependant and the last one depending on the server. The size of a Job Unit should try to minimize this equation. If you have a single core setting, e.g. using threads, the two communication variables could arguably be considered null, but having many threads will greatly increase the other two parameters.

2 Application Specific Information

In this application you have three distinct Distributable Job instances:

RepresentativesJob : Create a set of representatives (call it R) with the following properties:

- $\forall x, y \in R : x \neq y \Rightarrow distance(x, y) > cutoff$
- $\forall x \in Structures : \exists r \in R : distance(x, r) \leq cutoff$

ClustersJob : Create a partition on $Structures$, say $P \in Structures^N$ with the following properties:

- $\forall S \in P : !\exists r \in R : r \in S$
- $\forall r \in R : !\exists S \in P : r \in S$
- $\forall S \in P, s \in S : \exists r, r' \in R : r \in S \wedge r \neq r' \Rightarrow distance(s, r) \leq distance(s, r')$

CentersJob : From each cluster, assign the center to it as the closest structure to the geometric mean of the cluster. If we assume that this center is $c \in C$, where C is a particular cluster ($C \in P$), then we have:

- $\forall s \in C, c' \in Centers : c \neq c' \Rightarrow distance(s, c) \leq distance(s, c')$

In layer 1 you can also find the definition of a Structre (the atom backbone of a given protein) and all classes encompasing the handling of a Structre Data Base. That is, you can find classes handling the status of partially formed Structure Clusters and the Sets of Marked and UnMarked Data.

Outside the layers you can find alternate libraries implementing standard containers (used thourghout the application) or even application specific ones such as GROMACS.

Within this layer view, and specific to this application, it could be argued that layer 3 alone constitutes a framework to deal with BOINC clients, layers 2 and 3 taken together form a framework for generic problem solving using BOINC and all three layers taken as one single piece of software create the clustering application.