The `ClusteringSuite` package

12th March 2014

The `ClusteringSuite` is the piece of code that implements all the cluster analysis techniques developed at the Barcelona Supercomputing Center Tools Team. Basically, it is composed by two main libraries, `libClustering` and `libTraceClustering`, and a set of binaries that use these libraries to offer the different features. All this collection of software follows an object-oriented design and is implemented in C++, with limited features implemented in C.

1 Software engineering

In this section we present the a coarse-grain description of the software engineering behind the `ClusteringSuite` package. This package represents the third version of the software package that aggregates a set of features that became stable as the development of the thesis advanced.

The features of the package are divided in two main parts: first, a core cluster analysis library, `libClustering`, that includes the abstraction of the information containers and the clustering algorithms; second, the `libTraceClustering` that offers the features of extracting the required information from application traces, prepare the information to perform to use the `libClustering` and generate the different outputs. Both libraries offer a clean façade class to access the different features they implement that is used by the different binaries. In the following points we detail the classes that compose each library and the interaction among them.

1.1 libClustering

Figure 1 contains a basic UML class model of the `libClustering`. It contains the four main classes required to perform a cluster analysis. First, a *Point*, the abstraction of n-dimensional point including basic operation such as the Euclidean distance to another Point. A set of points is aggregated over a *DataSet*, useful

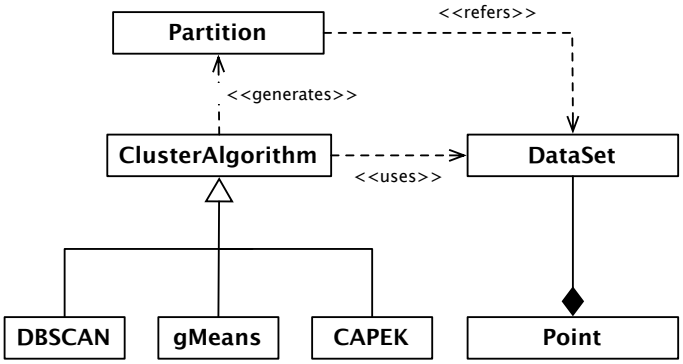


Figure 1: UML class model of the `libClustering` library

to manipulate data ranges or to build indexes to ease the access to each individual point. Next, the hierarchy where the top class is *ClusteringAlgorithm* acts as an interface to the actual implementations of multiple algorithms. The ClusteringAlgorithm objects process a DataSet and generate a *Partition*, a class that relates the Points in a DataSet to the cluster they belong. In this Figure we just depict three of the possible cluster algorithms this library offers.

1.2 libTraceClustering

`libTraceClustering` is the library that includes all the logic required to extract the information from an application trace (a Paraver trace or a Dimemas trace) and then process it to execute a cluster analysis using the `libClustering` library. As can be seen in the UML class model of this library, Figure 2, the interaction between these two libraries relies on a hierarchy inheritance, where *TraceDataSet* and *CPUBurst* are specializations of *DataSet* and *Point* respectively defined in the `libClustering` library.

ClusteringDefinition class contains the information to set up all the model. First, it defines which of the cluster algorithms from the `libClustering` will be used (and the possible values for its parameters). Second, it defines the different *ClusteringParameter* objects. Each ClusteringParameter represents one of the dimensions that describe a Point (specialized as CPUBurst at this level) used by the ClusteringAlgorithm to discover the different clusters.

Using the ClusteringParameter's, a *DataExtractor* object will read the contents of a *Trace* filling the TraceDataSet with the CPUBursts found.

Then the particular ClusteringAlgorithm is executed and creates the corresponding *Partition* object (also part of the `libClustering` library). This Partition object will be used first by a *ClusteringStatistics* object to compute statistics and by *PlotManager* (also defined by the ClusteringDefinition object) to generate

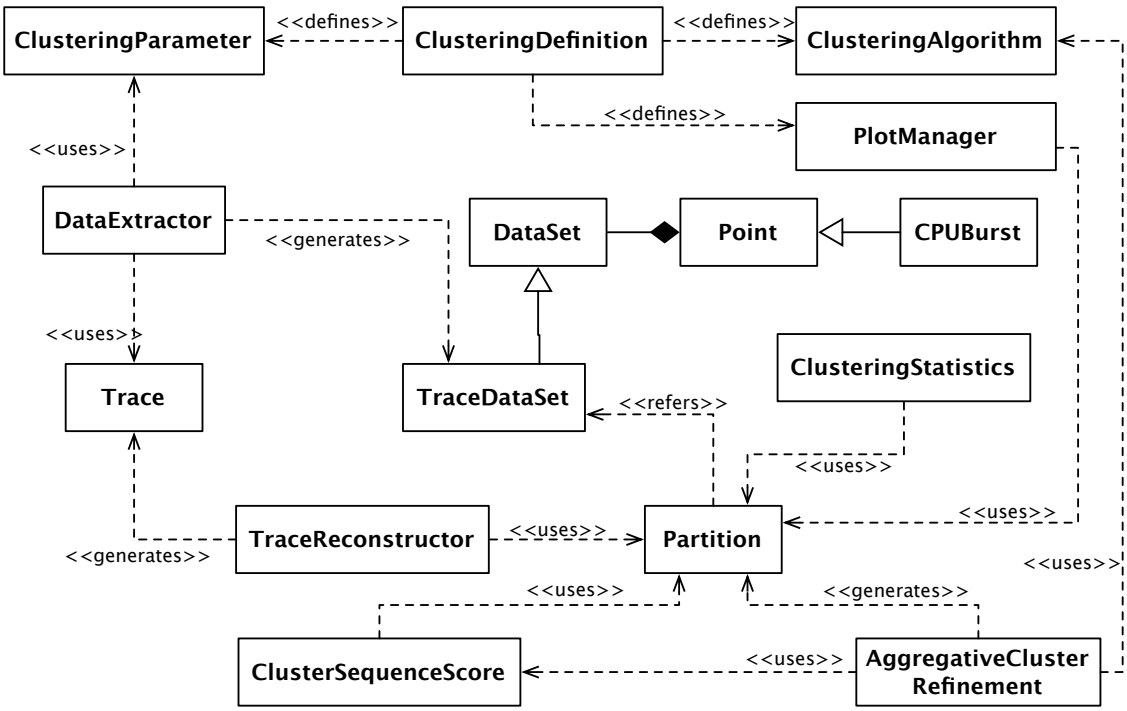


Figure 2: UML class model of the `libTraceClustering` library

output plots of the clusters found. Then *TraceReconstructor* will create an output trace with the same information of the original and the information (events) to identify to which cluster each CPU bursts belongs.

In case of using the Aggregative Cluster Refinement algorithm, the library behaves differently than using a regular cluster algorithm. Basically, it will make use of DBSCAN (to clarify the drawing, in Figure 2 it is represented as it uses any ClusteringAlgorithm) on each refinement step as well as a *ClusterSequenceScore* object to evaluate the quality of the intermediate clusters (the different Partitions). AggregativeClusterRefinement produces a final Partition of the data, but the library can keep track of intermediate ones to also produce intermediate traces and plots useful to evaluate the hierarchical generation of the final clusters.

2 Libraries and tools

The tools offered in the `ClusteringSuite` package are `BurstClustering`, `Clustering-DataExtractor` and `DBSCANParametersApproximation`.

`BurstClustering` The main tool that includes the cluster analysis based on a

application trace. The user provides an XML file (used by all three tools) to configure the analysis (a *ClusteringDefinition* class in the model) and a trace file. Using the `libTraceClustering` it processes the provided trace, run the cluster algorithm (both the ones implemented in the `libClusterig` or the Aggregative Cluster Refinement) and generates the output trace and the cluster statistics and plots.

`ClusteringDataExtractor` A tool that only offers the data extraction from the input trace and the plot generation, but not the cluster analysis. It is useful to performn preliminary observations about the data distribution so as to adapt the parameters used by the cluster analysis, for example to filter the individuals.

`DBSCANParametersApproximation` This tool is useful when using the DBSCAN algorithm to help the user to tune the algorithm parameters.

3 `ClusteringSuite` tools usage

This section is intended as a brief manual of regular use of the three tools included in the `ClusteringSuite` software[1]. As mentioned before, the tools offered in the software package use an input trace where the information is extracted. Even it could be a Paraver or Dimemas trace, it almost all cases, the input trace is a Paraver trace. The second input file these tools require is the configuration file XML. This file is key to define which the parameters of the clustering process.

In brief, the cluster analysis process its composed by 6 steps. First four steps are required to generate the XML configuration file, while the last two are the execution of the `BurstClustering` tool itself and the observation and analysis of the results.

1. Select of the clustering/extrapolation parameters.
2. Define the filters and normalization applied to the input data
3. Select the cluster algorithm and its parameters
4. Define the output plots
5. Execute the cluster analysis
6. Observe the different outputs

[1] All the guidelines presented in this section are applicable to the `ClusteringSuite v2.XX`

The actual definition of the different records in the XML file are described in the following section (4), while this one include the guidelines to detect the information it will contain.

1. Select the clustering parameters

The first decision to take when performing a cluster analysis is which of the data present in the input trace will be used to describe each CPU burst, in the `ClusteringSuite` terminology, we call them simply the *parameters*.

Using the Paraver vocabulary, a CPU burst is expressed in a trace as a *State Record* of value 1 (*Running State*). The parameters available to characterize a CPU burst are those events that appear at the end time of the given Running State. As a Paraver event is a pair event/value, in the XML file we use the event type to indicate events we whose values will be stored in the different bursts. We can also use Running State duration (difference between end time and begin time) as a CPU burst parameter.

In the XML we will express those parameters that will be used by the cluster algorithm, the *clustering parameters*, and those that will be used in the extrapolation process, the *extrapolation parameters*. The parameters can be defined as single event reads (*single events*) or combinations of pair of events (*mixed events*). In case we use the CPU burst duration, it will always be used as a clustering parameter.

It could be obvious, but to define the different parameters it is essential to know first which ones we want to use and which are the event type codification present in the trace. To do that we need to go through to the Paraver Configutarion File (*.pcf* file generated by `Extrae`) and check which events appear in the trace and their event type encoding. Almost in all analyses we use the Performance Hardware Counters events, being Completed Instructions and IPC the usual metrics combinations used by the cluster algorithm.

2. Define the filters and normalizations

Once knowing which are the clustering parameters, we have to decide the possible filters we want to apply. The filters prevent the cluster algorithm of analysing CPU bursts that can bias the result or do not add any valuable information. We found two different filters: a duration filter to discard those burst whose duration is shorter than a given value, and a range filter that can be defined to each parameter and eliminates those bursts than are out of the boundaries.

To tune the duration filter we use the `stats` tools provided by the CEPBA-Tools package. Using the `-bursts_histo` parameter this tool computes a plot as the one presented in Figure 3 for a given Paraver trace. This plot is an histogram

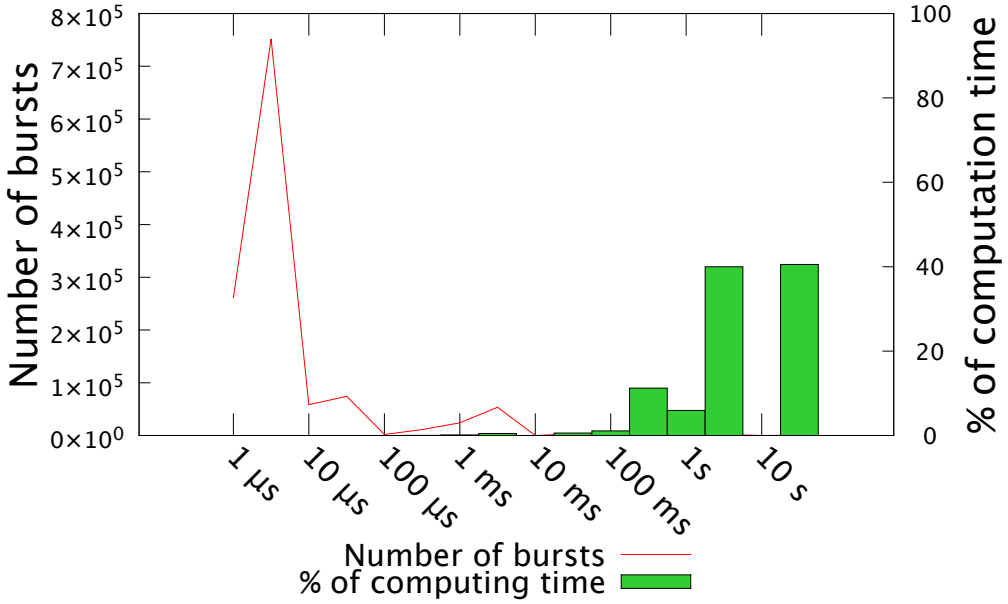


Figure 3: Bursts histogram produced by `stats` tool

where the x axis is the duration of the CPU bursts and quantifies both the aggregated time of the CPU bursts, the green bars, and the number of bursts, the red line. Observing this plot we can select the duration that eliminates de maximum number of bursts (red line at left of the select duration), while maintaining a high value of aggregated time (green bars at right of the selected point). For example, in the Figure 3, a reasonable duration filter will be 10 miliseconds.

With respect to the normalizations, we provide the possibility of applying first a logarithmic normalization, useful when the parameter range is wide and can bias the results of the cluster analysis. The logarithmic normalization can be applied to each parameter independently. The second normalization is a pure range normalization to set the parameter values in range $[0, 1]$, following the formula range ($\forall a_i \in A, a_i \leftarrow (a_i - min(A))/(max(A) - min(A))$). When using the range normalization, it will be applied to each parameter used, so as to guarantee that all of them have the same weight in the analysis. If we to add more weight one of the parameters used in the cluster analysis, we can apply a multiplicative factor.

To clarify how the different normalizations and filters work, this is the order as they are applied: when a CPU burst is read, its duration is checked and then the different parameters that have range filters defined; to those bursts that pass the filters its performed the logarithmic normalization of each parameter that requires it and afterwards the range normalization. Finally, the scaling factor is

Cluster Algorithm Name	Parameters
DBSCAN	epsilon, min_points
GMEANS	critical_value, max_clusters
CAPEK[1]	k
MUSTER_PAM[1]	max_clusters
MUSTER_XCLARA[1]	max_clusters

[1] libClustering includes a common interface to this algorithms offered by the MUSTER library (http://tgamblin.github.com/muster/main.html

Table 1: Cluster algorithms included in the libClustering and their parameters

applied.

3. Select the output plots

We can combine the parameters defined previous to generate GNUplot scripts of 2D and 3D scatter-plots. The plots can print both the normalized data or the raw data (before normalizations). The user can tune tune the ranges to print and also the axis-labels of the plots. In addition, users can let the library to produce all 2D plots obtained combining all metrics defined.

Once having the parameters, filters and plots, we can run the application ClusteringDataExtractor to extract the data and produce the plots described before runing the cluster algorithm. The resulting plots will show all the data available, distinguishing between the duration filtered bursts, the range filtered bursts and the ones that will take part in the cluster analyses. These plots are an useful aid to fine tune the parameter filters and normalizations.

4. Select the cluster algorithm

Even though the Aggregative Cluster Refinement and DBSCAN are the two basic algorithms offered by the ClusteringSuite package, there is a few more clustering algorithms offered to the user. Table 1 contains the list of these algorithms and their parameters. It is interesting to note that the Aggregative Cluster Refinement is the only algorithm that does not require any parameter and it not have to be expressed in the XML configuration file.

For further information about the different algorithms included in the package, we point to the following papers: [1] for DBSCAN algorithm, [2] for GMEANS

and [3] for CAPEK, PAM and XCLARA.

In case of DBSCAN we provide the application `DBSCANParametersApproximation` to help the paremeter selection, according to the technique described in [1].

5. Execute the cluster analysis

Once defined the different elements necessary to perform the analysis, we need to execute the `BurstClustering` tool. The different parameters of this tool and a short description of them are listed in Table 2. Basically, to perform a regular analysis using the cluster algorithm defined in the XML file we need to execute the command:

```
BurstClustering -d <clustering_definition.xml> -i <input_trace> -o <output_trace>
```

The tool will process the information provided in the configuration file, extract the data from the input trace, execute the cluster algorithm and then generate the required output plots, extrapolation files and the output trace. These files will be explained in the further step.

In case we want to execute the Aggregative Cluster Refinement algorithm, the command varies slightly:

```
BurstClustering -d <clustering_def.xml> -ra[p] -i <input_trace> -o <output_trace>
```

By adding the `-ra` parameter, the tool discards the algorithm indicated in the clustering definition XML file and then applies this different algorithm. In case we use the parameter `-rap`, the tool will produce, apart from the regular outputs, the traces and plots of intermediate steps of the Aggregative Cluster Refinement algorithm.

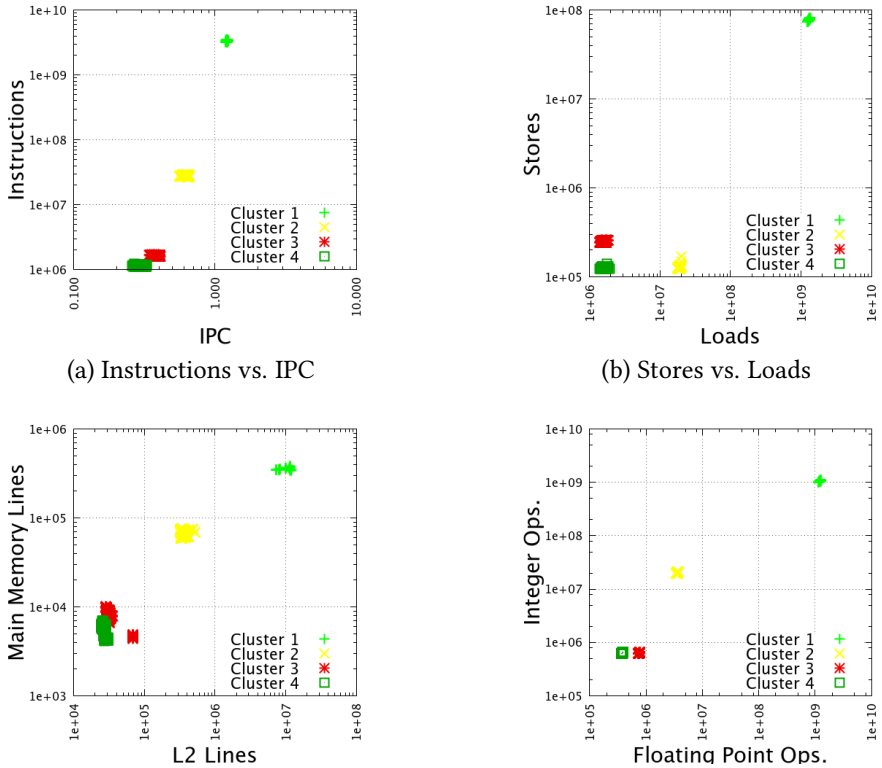
6. `BurstClustering` tool outputs

The `BurstClustering` offers three main outputs: scatter-plots of the different metrics, a cluster statistics file (including the extrapolation) and a reconstructed Paraver trace. In addition, it also generates the refinement tree, when using the Aggregative Cluster Refinement. Optionally, it can produce the a file with the sequence alignment and a file containing the Cluster Sequence Score values. Here we will describe briefly all of them.

The **scatter-plots** are simply GNUplot scripts that can be load using this plotting tool. As seen in previous steps, they can be 2 or 3 dimensional combinations of different metrics used to characterize the CPU bursts. In any case, the points in the scatter plots are coloured to distinguish the different clusters found. These plots are useful to observe, qualitatively, variations in the clusters

Parameter	Description
Required parameters	
`-d <clustering_definition.xml>`	Clustering definition XML to be used
`-i <input_trace.prv>`	Input (Paraver) trace to be used
`-o <output_trace.prv>`	Output (Paraver) trace with cluster information added
Optional parameters	
`-h`	Show help message and exit
`-s`	Performs a silent execution, without informative messages
`-m <number_of_samples>`	Performs a cluster analysis using the number of burst indicated and classifying the rest
`-a[f]`	Generates an output file with the aligned sequences (in FASTA format when using '`-af`')
`-ra[p]`	Executes the Aggregative Cluster Refinement instead of the cluster algorithm indicated in the XML. When using '`-rap`' generates plots and outputs from intermediate steps
`-t`	Print accurate timings (in $\mu seconds$) of different algorithm parts
`-e EvtType1, EvtType2,...`	Changes the Paraver trace processing, to capture information by the events defined instead of CPU bursts

Table 2: `BurstClustering` tool parameters



(a) Instructions vs. IPC

(b) Stores vs. Loads

(c) Main memory accesses vs. L2 data cache acessess

(d) Integer instructions vs. Floating point instructions

Figure 4: Output plots produced by `BurstClustering` tool combining different metrics

with respect to the metrics used. In Figure 4 we show 4 different plots combining 8 different hardware counters. First plot, 4a, show the metrics used by the cluster algorithm. In the rest of combinations we can observe that the clusters represent clear isolated clouds, with a minor exception of the plot comparing Main Memory Accesses vs. L2 Data Cache Accesses, 4c, where Cluster 4 (in red) appear in two different clouds.

The plot scripts are named using the output trace prefix plus a trailing string expressing the combination of metrics used. They have the extension `.gnuplot`. All of them use a file ended in `.DATA.csv` that contain on each line the different parameters described in the XML file plus the cluster identifier assigned for each CPU burst analysed.

The **clustering statistics** file is a CSV file that contains the number of individuals, the aggregated duration and the average duration per CPU burst, and the average values of extrapolation parameters defined in XML, for each cluster

found. This file is really useful to analysed quantitatively the behaviour of the different clusters found. The clusters statistics file is named using the prefix of the input trace, but ending in `.clusters_info.csv`.

Next output that is always produced is the **output trace**. Basically, this is exactly the same input trace where all the CPU burst have been surrounded using a certain events to identify them. Thanks to these events, we can take advantage of the vast analysis power of Paraver and Dimemas to perform further analyses and correlate the clusters with all the information present on the trace. For example, we can observe the time-line distribution of the different computation regions detected. An example of Paraver time-line and its corresponding duration profile can be seen in Figure 5. We provide a set of Paraver configuration files with pre-defined views and histograms related to cluster events.

In case we executed the Aggregative Cluster Refinement algorithm, the tool will also produce a **refinement tree** file. This file has the same prefix as the output trace and the extension `TREE.dot`. It is a text file that describes the refinement tree using the DOT language. To visualize it we require the GraphViz[2] software package. We also recommend using of the interactive tool `xdot`[3] to navigate through the refinement tree output. An example of a refinement tree can be seen in Figure 6.

Finally, using the parameter `-a`, the tool will produce a CSV file containing the sequences obtained after applying the Cluster Sequence Score. This file, named as the output trace with the extension `.seq`, contains the sequence of the cluster identifiers (numbers) and gaps (marked as hyphens) introduced by the alignment algorithm for each task and thread present on the input trace. If use the parameter `-af`, the file will be generated in the FASTA format, transforming the first 21 clusters in an amino-acid identifier. The FASTA file can be load in any alignment software, such as ClustalX[4] for its visualization. In Figure 7 we can see a ClustalX window with a set of aligned sequences.

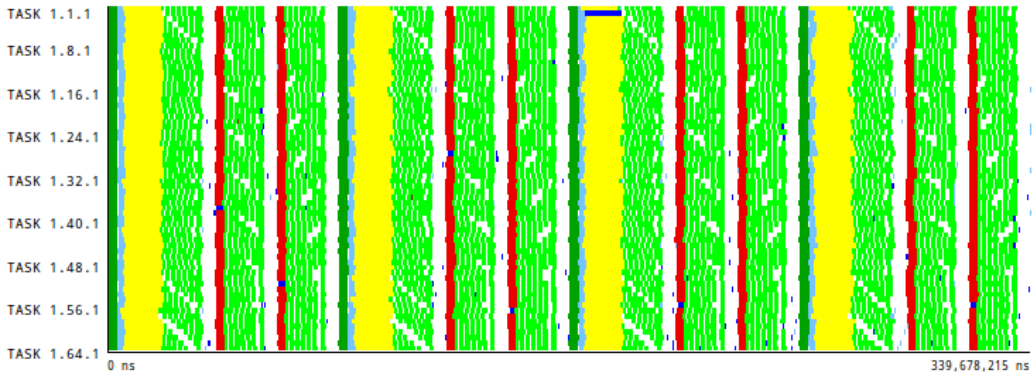
If we use any of these two last parameters, the tool will also produce a file with the extension `SCORES.csv`, that contains the numerical results of the Cluster Sequence Score.

When using the Aggregative Cluster Refinement with the parameter `-rap` the tool will produce the plots, traces, refinement trees, sequence files and score files for each refinement step. The intermediate statistics files will not be generated and these intermediate trace files will only contain cluster events, to check the intermediate cluster distribution, but not to correlate them with other information. The intermediate files will have an inter-fix `STEPX` in their file name, to

[2] `http://www.graphviz.org/`

[3] `http://code.google.com/p/jrfonseca/wiki/XDot`

[4] `http://www.clustal.org/`



(a) Time-line distribution of discovered clusters

	Cluster 1	Cluster 2	Cluster 3	Cluster 4
TASK 1.1.1	103,371,000 ns	53,163,000 ns	16,951,000 ns	11,573,000 ns
	101,279,000 ns	40,256,000 ns	19,191,000 ns	11,871,000 ns
	108,464,000 ns	54,570,000 ns	19,586,000 ns	11,895,000 ns
	108,587,000 ns	54,543,000 ns	19,852,000 ns	11,898,000 ns
	106,993,000 ns	54,539,000 ns	19,700,000 ns	11,818,000 ns
	108,652,000 ns	54,529,000 ns	19,631,000 ns	11,850,000 ns
	95,878,000 ns	54,660,000 ns	19,683,000 ns	11,892,000 ns
TASK 1.8.1	101,991,000 ns	53,945,000 ns	17,353,000 ns	11,490,000 ns
	96,053,000 ns	53,805,000 ns	20,579,000 ns	12,015,000 ns
	103,801,000 ns	53,615,000 ns	21,393,000 ns	12,348,000 ns
	109,804,000 ns	54,624,000 ns	21,986,000 ns	12,328,000 ns
	110,764,000 ns	54,577,000 ns	21,927,000 ns	12,368,000 ns
	111,561,000 ns	54,528,000 ns	21,877,000 ns	12,269,000 ns
	100,226,000 ns	54,607,000 ns	22,040,000 ns	12,320,000 ns
	96,208,000 ns	53,757,000 ns	20,143,000 ns	12,266,000 ns
TASK 1.16.1	100,976,000 ns	53,729,000 ns	19,193,000 ns	11,916,000 ns
	95,966,000 ns	53,682,000 ns	20,646,000 ns	11,920,000 ns
	104,536,000 ns	53,613,000 ns	21,416,000 ns	12,313,000 ns
	110,702,000 ns	54,448,000 ns	21,805,000 ns	12,220,000 ns
	109,171,000 ns	54,436,000 ns	22,099,000 ns	12,275,000 ns

Cluster 1

(b) Duration histogram of the clusters per application task

Figure 5: A Paraver time-line and profile showing information related to a cluster analysis

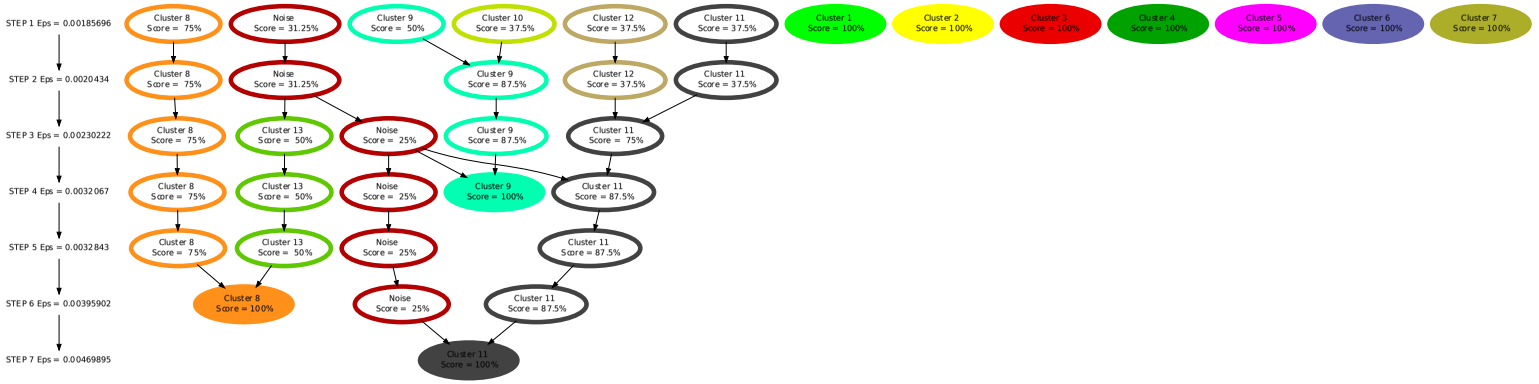


Figure 6: Example of a refinement tree produced by `BurstClustering` tool

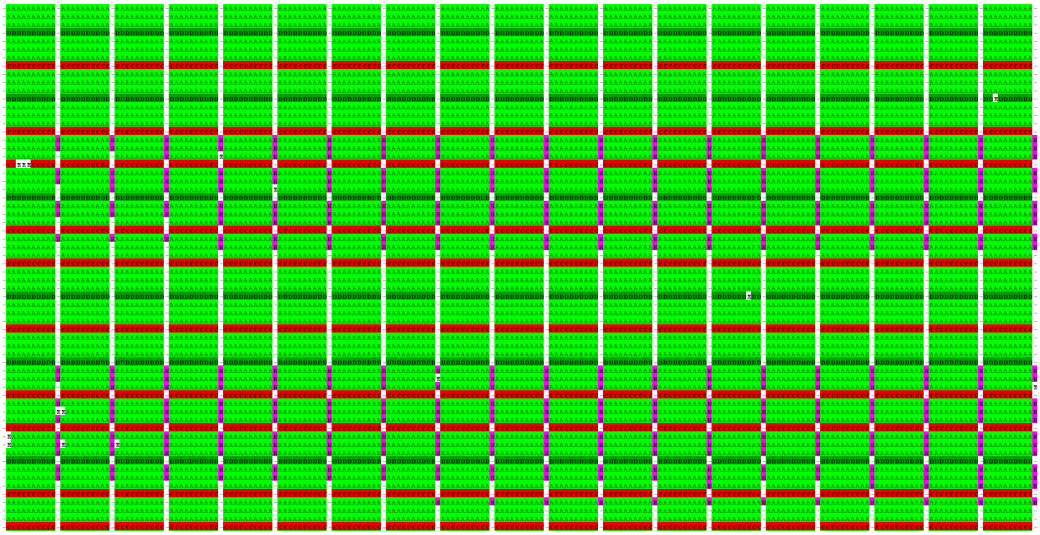


Figure 7: ClustalX sequence alignment window

distinguish at which step (iteration) of refinement were produced.

Finally, it is interesting to note that we guarantee the colour coherence in all those outputs generated by the `BurstClustering` that use colour information to distinguish the cluster identifiers. In case of ClustalX we provide a modified version of software package with the required amino-acid colouring.

4 Creating the clustering definition XML

In brief, the clustering definition XML file contains the description of four elements of the clustering process: the parameters associated to each CPU burst in the trace used by cluster analysis and the extrapolation process; the filtering ranges and normalizations applied to this data; the cluster algorithm to be used; and finally, the description of the different output plots, generated as GNUplot scripts. We can see how these different parts are distributed in the XML file in Figure 8.

Following the current description of the file it could be easily generated using a regular text editor or a XML editor.

4.0.1 Parameter selection

There are two ways to define how the parameters are read from a Paraver trace. First, the values of individual events situated at the end of the Running State, using `single_event` nodes. Second, combining the values of two different events with a basic mathematical operation, using `mixed_events` nodes.

```
<clustering_definition use_duration="no" apply_log="yes" normalize_data="yes"
duration_filter="10000" threshold_filter="0">

  <clustering_algorithm name="xxx">
    <!-- specific cluster algorithm parameters -->
  </clustering_algorithm>                          Cluster algorithm parametrization

  <clustering_parameters>
    <!-- 'single_event'/'mixed_events' nodes defining the CPU bursts
         parameters (dimensions) to be used by the cluster algorithm -->
  </clustering_parameters>      CPU bursts parameters to be used by cluster algorithm

  <extrapolation_parameters>
    <!-- 'single_event'/'mixed_events' nodes defining the CPU burst
         parameters (dimensions) to be used in the data extrapolation
         process -->
  </extrapolation_parameters>              CPU bursts parameters to be extrapolated

  <output_plots all_plots="no">
    <!-- 'plot_definition' nodes defining 2D/3D plots combining the
          'clustering_parameters'/'extrapolation_parameters' --
  </output_plots>         Definition of the output plots combining the CPU parameters

</clustering_definition>                                                  End tag
```

Figure 8: Clustering definition XML file structure

A **`single_event`** node, see Figure 9a, contains first two attributes: `apply_log` that indicates if a logarithmic normalization will be applied to its values; the `name` parameter is the label the will be used in the different output file. The inner node `event_type` is mandatory, to define the event type that appears in the Paraver trace. Optional nodes `range_min` and `range_max` are used to filter the CPU burst outside these boundaries. Finally, optional node `factor` is a multiplicative value so as to weight the parameter value.

A **`mixed_events`** node, see Figure 9b, is pretty similar to the previous one, but includes two mandatory internal nodes `event_type_a` and `event_type_b`, to define the two types of events involved, and the attribute `operation` to define the mathematical operation applied to the values read. Possible operations are `+`, `-`, `*` and `/`. The operation is applied to the values of the two events defined, *before* the logarithmic normalization.

To define the CPU bursts parameters that will be used by the cluster algorithm, they have to be placed below the `clustering_parameters` node, see Figure 8. To define those that will be used to characterize the resulting clusters (as averages in the `.clusters_info.csv` file), we have to place them below the `extrapolation_parameters` node.

If we want to use the duration of the CPU bursts as a parameter, we need to set

```
<single_event apply_log="yes" name="PM_INST_CMPL">
  <event_type>42001090</event_type>
  <range_min>1e6</range_min>
  <range_max>1e8</range_max>
  <factor>1.0</factor>
</single_event>
```

(a) `single_event` node structure

```
<mixed_events apply_log="yes" name="IPC" operation="/">
  <event_type_a>42001090</event_type_a>
  <event_type_b>42001008</event_type_b>
  <range_max>3</range_max>
  <factor>1.0</factor>
</mixed_events>
```

(b) `mixed_events` node structure

Figure 9: Nodes to define the parameters extracted from a trace

to *yes* the attribute `use_duration` present in the root node (`clustering_definition`).

4.0.2 Filtering and normalization

The filtering and normalization is expressed at two points of the XML file. We have seen that the parameter definition nodes include both a range filtering and also a logarithmic normalization. The filtering information included in the extrapolation parameters is not taken into account.

The second point is the root node. In this node we find different attributes, see Figure 8 regarding filters and normalizations. First one is `apply_log`, that indicates if logarithmic normalization will be applied to the burst duration, if used. Next one is `normalize_data`, that indicates if a final range normalization will be applied to the values of *all* parameters (independently). Next we find the `duration_filter`, expressed in μs, to discard those burst with less duration than the indicated. Finally, the `threshold_filter` is a percentage to discard all the clusters found whose aggregated duration represents less percentage of the total clusters duration than the indicated.

4.0.3 Output plots

Once defined the parameters used to characterize the CPU bursts, below the `output_plots` node we can define the output plots combining the different metrics.

If we set the attribute `all_plots` of this main node to *yes*, the `libTraceClustering` library will generate all possible 2D plots combining the parameters defined (clustering parameters and extrapolation parameters). If we want to

```
<plot_definition raw_metrics="yes">
  <x_metric title="IPC" min="0.1" max="2">IPC</x_metric>
  <y_metric title="Instr. Completed" min="4e7" max="5e7">PAPI_TOT_INS</y_metric>
  <z_metric title="Memory Instructions">Memory_Instructions</z_metric>
</plot_definition>
```

Figure 10: `plot_definition` node of the clustering defintion XML

manually define the combinations we can use the `plot_definition` structure, see Figure 10.

What we find first in the `plot_definition` node is the attribute `raw_metrics`. In case we applied normalization to the clustering parameters setting this attribute to "yes" indicates that the resulting plot will use the raw values of the parameters. Then we find three kind of nodes `[x|y|z]_metric`. Each of these nodes has a mandatory attribute `title` that will be used as the plot label for the corresponding axis. They have two optional attributes `max` and `min` to define the axis range. Finally, the content of each of these nodes must be the *name* attribute of any of the parameters defined previously (clustering parameter of extrapolation parameter). In case we want to use the duration, as it is defined differently from regular parameters, it has to be referenced simply using the text `Duration`.

We can combine up to three metrics to create a 3 dimensional scatter-plot, where the individuals will be distinguished in series according to the cluster identifier assigned. The same is applicable when using just two metrics (x and y). If we just define a single metric (*x metric*), the resulting plot will be a 2 dimensional plot using the cluster identifier as y axis.

References

[1] M. Ester, Hans P. Kriegel, J. Sander, and X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In Evangelos Simoudis, Jiawei Han, and Usama Fayyad, editors, *KDD96: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231, Portland, Oregon, 1996. AAAI Press.

[2] Aislan Gomide Foina, Rosa M. Badia, and Javier Ramirez Fernandez. G-Means Improved for Cell BE Environment. In *Facing the Multicore-Challenge*, pages 54–65, 2010.

[3] Todd Gamblin, Bronis R. de Supinski, Martin Schulz, Rob Fowler, and Daniel A. Reed. Clustering Performance Data Efficiently at Massive Scales. In *ICS '10: Proceedings of the 24th International Conference on Supercomputing*, pages 243–252, Tsukuba, Japan, 2010. ACM.