Authors: Gabriel de Souza Seibel, Gerônimo Veit Acosta, Vilmar A. Fonseca Filho.
Professor: Paolo Rech
Fundamentals of Fault-Tolerance - UFRGS (Universidade Federal do Rio Grande do Sul)
03 July 2018

Fault Tolerance in Clustering Algorithms

Introduction

The purpose of this document is to expose the results of the fault-injection experiments done using CAROL-FI [1] over three clustering algorithms: *k*-means, DIANA (Divisive Analysis) and AGNES (Agglomerative Nesting). CAROL-FI is a fault injector designed to provide its users understanding on programs' vulnerabilities to faults. These specific algorithms and injector were chosen so that we could compare PVFs (Program Vulnerability Factors) of applications that are used in a trending field of computer science: machine learning.

There are some implementation choices that we can make when working with clustering algorithms, such as the definition of distance between points and the linkage between clusters. We made use of the euclidean distance definition, coupled with single linkage between groups of points. Single linkage means that the distance from a group *A* to a group *B* is the smallest distance between points of pairs <*pA*, *pB*>, where *pA* is a point from group *A* and *pB* one from group *B*.

The most basic of the three algorithms is *k*-means. First, it initializes clusters centroids at random locations. Afterwards, it repeatedly catalogs what cluster each point belongs to, and recalculates the centroids of each cluster to be a point that has its attributes set to the mean attribute values of its points. All executions of *k*-means had 100.000 points as input, each with 34 integer values as attributes, and clustered them in 5 groups.

The other two clustering techniques are of the Hierarchical Clustering type. DIANA and AGNES differ by its approach - the former uses Top-Down clustering while the latter makes it

Bottom-Up. This means that DIANA repeatedly splits clusters into two new smaller ones (using *k*-means), while AGNES agglomerates points to clusters one at a time. Both output a dendrogram, which is essentially a binary tree of clusters, so that the user can choose a tree level to cut the dendrogram, getting the desired number of clusters. DIANA received as input 20.000 points with 10 floating points attributes each, while AGNES got only 2000 points with 2 attributes each. This decision of unbalancing the input size was made considering the slower execution time of this implementation of AGNES compared to the DIANA one, so that the faults could be injected in time.

The whole process of fault-injection was made using a simple personal computer running Linux Mint 18.3 Cinnamon 64-bit - with a Linux kernel of version 4.10.0-38-generic and GDB 7.11.1. The machine is equipped with an Intel Core i7 4770K processor and 16GB of RAM. The GPU model of the computer didn't matter, since the experiments ran on CAROL-FI, and not on a GPU fault-injector like SASSIFI. We imagine that a future study comparing AVFs for these clustering applications in GPUs could also be very fruitful for the areas of fault-tolerance and machine learning, though the study documented in this paper is scoped towards x86 architectures only.

We abstract the AVF (Architectural Vulnerability Factor) to what we can actually measure using CAROL-FI - the PVF - ignoring the HVF [2] (Hardware Vulnerability Factor) of the x86 architecture, since we are making use of such high-level fault-injector. In other words, the HVF is constant and unknown on our measures, and we can focus our efforts on estimating the PVF to have a reasonable basis for AVF analysis.

SDC Criticality

We believe that a critical SDC on the DIANA algorithm wouldn't be one that simply swaps some points between clusters, but one that disrupts the chaining of the clusters. This is because we store the various clusters of a dendrogram level in linked lists, and the clusters pointers are mapped to integer IDs (for plotting software like *graphviz* to generate graphs from it). SDCs that change this IDs are critical for our results, because it would change the whole tree structure of the dendrogram, making plotting of the structure problematic for human studies (which is the main goal of these clustering algorithms). That being said, we can pinpoint a variable that would cause this kind of havoc to the computation: the cluster-pointer to integer map variable "*ids*"; that stores the references mapping to IDs just mentioned.

For *k-means*, if we have a fault when the algorithm calculates the new position for the center, this could implicate in a critical SDC. For example, if the computation assigns a position to the center that is very far from the points, we should have a centroid that converges to a place that makes no sense to a human analysis. In this case, our output would be completely wrong, but only noticed by a secondary analysis.

Finally, A fault injection on AGNES would probably originate a critical SDC if it occured right when the execution was calculating the euclidean distance between two clusters. If an incorrect number is originated from this injection the new root cluster will be wrongly set and the whole hierarchical logic of tree of clusters would have great chances to be compromised. This process would generate a complete different output from what would be expected. Since this agglomerative algorithm depends on the creation of new roots in every level this change is highly propagative if it occurs in first steps of the tree generation. Another critical SDC can occur while injecting a fault in the index of the node since it abstracts the whole structure of the cluster into one integer number. If this number is changed the process would get lost while trying to track the correct information of the cluster hierarchy.

Unhardened Code

Fault Injection Results - Unhardened Code

The results of interest in this study are the SDCs (Silent Data Corruptions) and the DUEs (Detected Unrecoverable Errors), together with the masked faults, so that the PVFs can be estimated. For that, over 3000 faults were injected into each algorithm's execution, though we were also attentive to produce more than 1000 SDCs for each.

The chart in Figure 1 shows the fault-injection results (SDCs, DUEs and Masked Errors) in absolute values. Considering that the number of injected faults is not the same for all three applications, in Figure 2 we display the results above cited in percentage. From these primary values obtained from injections, we could then calculate the PVF for SDCs and the PVF for DUEs of each program, as shown in Figure 3.

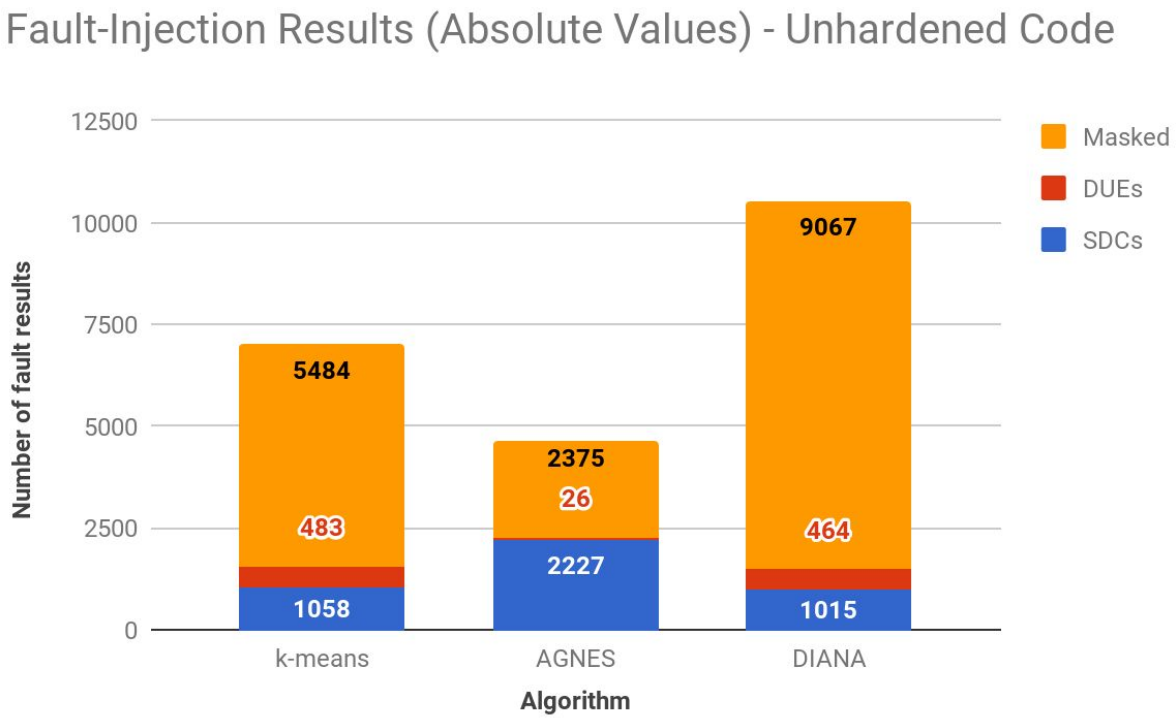

Figure 1 - Number of SDCs, DUEs and Masked for each unhardened algorithm

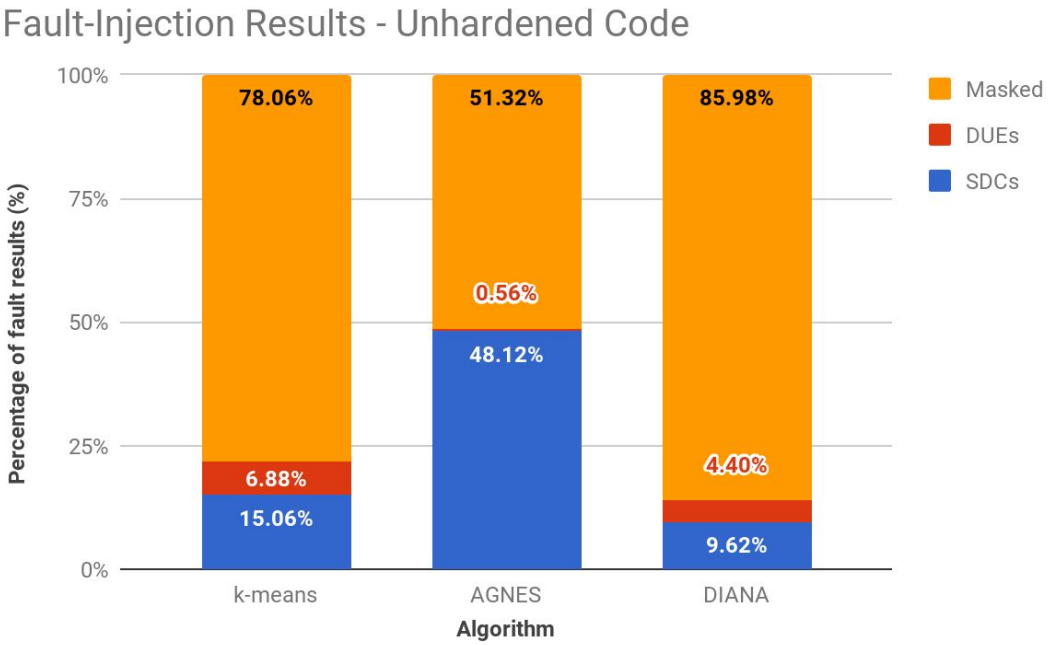

Figure 2 - Percentages of SDCs, DUEs and Masked for each algorithm - Unhardened Code

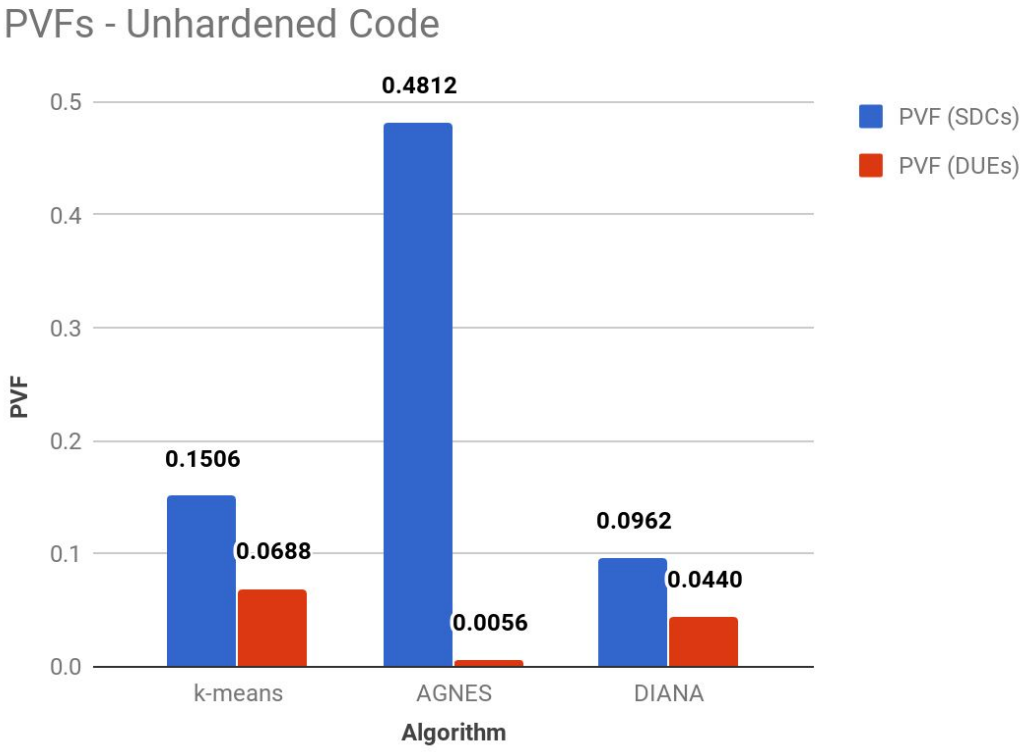

Figure 3 - The estimated values of PVF for each clustering technique - Unhardened Code

At first glance, the charts of Figure 1 and Figure 2 show three behavior peculiarities:

(i) The studied algorithms (considering its specific implementations) are much more sensible to SDCs than to DUEs.

(ii) DIANA and *k*-means have a very similar vulnerability behavior for SDCs and DUEs - both in magnitude and proportion.

(iii) AGNES has the highest PVF for SDCs of the three techniques, but also the lowest PVF for DUEs.

We can relate observation (i) to results seen in the paper Experimental and Analytical Study of Xeon Phi Reliability [1], in which faults were injected in HPC systems. This study showed consistently higher counts of SDCs than DUEs for the tested applications (safe one of the five).

Observation (ii) can be explained by the fact that DIANA makes heavy use of *k*-means in its computations, while adding complexity, since it creates a dendrogram - e.g. a tree of clusters - instead of *k* clusters. The added complexity also explains why PVFs for both SDCs and DUEs are slightly higher for DIANA than for *k*-means.

Finally, (iii) goes to show that AGNES is the most sensible of the three for data corruptions, but is roughly ten times more robust for hangs and crashes than the other ones. This is probably in a small part due to the peculiarities of the utilized implementation, since the code used for it is from a different author than the one from *k*-means and DIANA (although written by us) makes use of the routines *k*-means main clustering routine. That said, we believe that the most influential reason for these results is that the ideal AGNES algorithm (not accounting for implementation specificities) is actually more prone to SDCs than the others. We motivate this claim given the huge difference in data corruption counts that can't be accounted for just by the implementation-specific approach to the algorithm, and also because AGNES and DIANA use wholly different paradigms to clustering. Though both provide clustering outputs in the same (ideal) data structure, the dendrogram, the two cluster data of each group of the dendrogram levels using different techniques (agglomeration and *k*-means splitting).

Time Window Analysis - Unhardened Code

Figure 4 - The percentage of SDCs produced in each time window - Unhardened Code

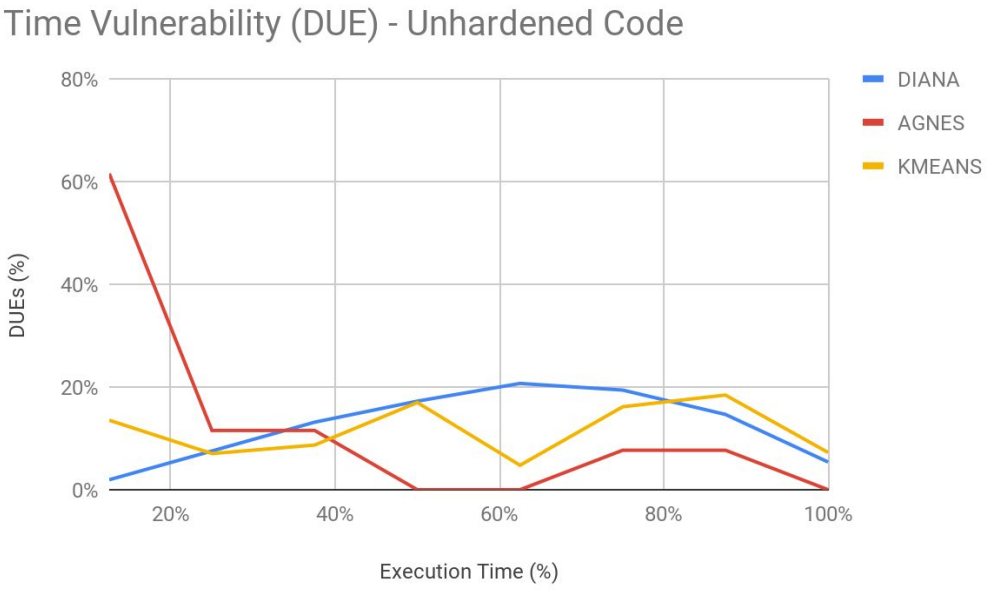

Figure 5 - The percentage of DUEs produced in each time window - Unhardened Code

The curves on Figure 4 represent the percentages of SDCs caused by faults in each time-window (which were defined to be steps of 12.5% of the total execution time). Let us first analyse the blue curve, that represents DIANA's behavior. As we can see, it has a noticeable peak of vulnerability in about 65% of the execution time, which can also be seen (although less pronounced) in Figure 5, in which the DUEs time vulnerability is plotted. We credit this high levels of SDCs and DUEs on this time-window to the fact that this may be the time when the algorithm processes the levels with the highest number of clusters inside. This makes sense because the number of clusters per level increases as the processing gets to it's mid-time, but it decreases after clusters start holding single points, and can't be divided anymore - resulting in fewer clusters in the next level, in an accelerating pace.

The number of clusters in a level influences both the number of SDCs and the number of DUEs because it effectively enlarges a critical for-loop index variable called *cluster_to_divide_index*, as we notice from cross-referencing this time-window analysis with the critical variables list for DIANA (presented later in this document). This increase in value on *cluster_to_divide_index*, in turn, makes the dependance of the computation on this variable greater, since the loop controlled by it will run more times, resulting in more uses of the variable inside the loop, before it is eventually reset to 0 when the loop finishes and runs from the start again (the loop is inside another loop!).

The agglomerative nesting algorithm (AGNES), constructs a tree with a bottom-up approach, by processing the input elements, creating clusters and joining them together, building a hierarchy. The output file is a dendrogram that represents this tree of clusters. With that in mind, a fault injection in the beginning of the execution will probably represent incorrect values between neighbors clusters, making it impossible to correctly merge them to create the next root cluster in the hierarchy. This action would compromise all the next steps of the tree construction, so an SDC/DUE (in case of bit-flipped pointers) has high chances to occur as we can observe in the graphs.

Another section of the algorithm that has a lot of SDCs due to fault injections is the end of the execution. At this point, the process writes the cluster hierarchy as a dendrogram in the output file. A wrong information in this file would cause an SDC, but probably not a critical one,

since it would not affect the tree structure (since it was already built), only its final representation.

Finally, considering time vulnerability graph for *k-means*, we can see three peaks on the execution of the algorithm. The first one can be explained by the algorithm trying to determine how many attributes per cluster centers it will utilize - if it reads the input in a wrong way, the whole output would be compromised. We believe that the two other peaks are just oscillations from a mean line that, in the case of *k-means*, doesn't grow much, since the operations made throughout the final time windows is always the same. The DUE's are almost constant, due to the constant number of arrays and pointers. Since the number of pointers doesn't change, the number of possible invalid memory accesses doesn't grow, so it doesn't increase the DUE occurrency

Critical Variables - Unhardened Code

Variables Sorted By SDC-PVF

AGNES			
PVF	**#SDCs**	**Variable**	**Type**
100	134	i	int
100	56	root_clusters_seen	int
100	54	i	int
100	50	index	int
100	43	target	int

Table 1.1 - The most critical variables for SDCs in AGNES - Unhardened Code

DIANA			
PVF	**#SDCs**	**Variable**	**Type**
77.02	191	cluster_to_divide_index	int
64.71	11	j	int
55.38	103	i	int
52.73	116	n_features	int
39.13	81	cluster_index	int
36.65	92	n_clusters_in_anterior_level	int

Table 1.2 - The most critical variables for SDCs in DIANA- Unhardened Code

k-means			
PVF	**#SDCs**	**Variable**	**Type**
92.59	75	numAttributes	int
78.08	57	nclusters	int
53.52	38	loop	int
50.72	35	new_centers_len	int
44.82	78	pt1	float

Table 1.3 - The most critical variables for SDCs in k-means - Unhardened Code

The above table of critical variables for AGNES confirms some assumptions previously made, starting by the criticality of the variables that represents the identifier of each cluster - such as *index* and *target*. The second variable is only a mask used for index. Both have the same function in the code and if one of them is not correct the execution will not be able the perform the tree generation properly. This would probably generate a completely different dendrogram, since the identifiers of each cluster do not correspond to the expected ones, and the algorithm will not do the correct associations to generate the new root cluster. Another critical variable that is worth to be mentioned is the loop control variable - that is also used as an index to add the new leaves of each cluster that will be the new root for the previous cluster. This action will inhibit the program to compute the cluster hierarchy correctly, since the information in the root is corrupted with incorrect items (incorrect input elements).

Analysing the vulnerable variables list for DIANA, we see that the most critical ones are all for-loop indices or stop conditions. If one of the indices related to cluster counts gets its value swapped for something incorrect, the loop's inner operations won't access the corrects indices in the lists of clusters of each level, resulting in some computations being skipped or repeated, and the end result dendrogram getting a different number of clusters than it should.

The fourth most critical variable for SDCs in DIANA is *n_features*, a counter for the number of dimensions of each clustered data point. If this number is lower than it should be, this would make the algorithm ignore part of the input data, making the output most probably wrong.

As for the variable *n_clusters_in_anterior_level*, a for-loop stop condition, its sensibility can be explained by the fact that it controls quantities of linked-list chained accesses - better said, if its value is lesser than the expected, there will be, again, computations being skipped, resulting in an il-formed dendrogram.

In the table for *k-means*, we can see that the most critical variables are the ones that control the quantity of attributes and clusters and the variable "loop" that is used as a control variable to unallocate all the centers used in the clustering. The first two are obviously critical: the first is the number of attributes of the data points - when altered it can confuse the algorithm and causes inconsistencies with the calculated distance. The other one is the number of clusters, that can cause the same problem as before. This two variables are the main core of the algorithm, they are used at every new recalculation of distances in the application, this way, if the fault is injected in these variables, it will almost certainly cause errors on the output.

Variables Sorted By DUE-PVF

DIANA			
PVF	**#SDCs**	**Variable**	**Type**
58.43	104	num_omp_threads	int
19.35	48	cluster_to_divide_index	int
13.24	36	level	int
10.71	27	threshold	double
10.45	23	n_features	int

Table 2.1 - The most critical variables for DUEs in DIANA - Unhardened Code

k-means			
PVF	**#SDCs**	**Variable**	**Type**
73.17	60	nloops	int
36.7	29	num_omp_threads	int
23.19	16	out_filename	char
15.54	23	cluster_centres	float
15.38	24	partial_new_centers	float

Table 2.2 - The most critical variables for DUEs in k-means - Unhardened Code

The lack of information of AGNES in this section is due to the extremely low number of DUEs detected during the fault injection. In fact, the most relevant data was not related to our source code but to some external libraries detected by the gdb. The only variables in our source code that actually caused a DUE had the PVF really low (less than 1%). This fact did not confirm what we expected since we thought that there were great chances to a DUE occur if a fault was injected directly in a cluster pointer.

Interpreting the table of DUE-critical variables for DIANA, we identify the first one (*num_omp_threads*) as highly critical because it controls the number of elements in some

heap-allocated arrays. Consequently, if this number gets corrupted and gets higher than it should be, there will be invalid memory accesses, causing crashes. This variable actually belongs to a *k-means* function, used by DIANA. *Threshold* is another variable that influences the execution of the *k-means* calls inside DIANA, mainly because changes to it will cause clusters to be partitioned inappropriately.

We see another appearance of the variable *cluster_to_divide_index* in this table, as it had already showed up in the list of SDC-critical variables. The problem now is if this variable gets corrupted with a value *higher* than it should be, which would cause the algorithm to try to access fields of *null* variables, causing crashes. The same analogy can be made for the reappearing variable *n_features* and the variable *level*: the DUEs occur when this variable gets a higher value than the original one, because of invalid memory accesses.

We notice that *k-means* give us DUE's in variables that are used for loop control, variables that set the number of parallel threads that should work on that particular portion of the code, and variables that control the quantity of clusters centers. The variables that control loops can easily cause a DUE if faults are injected in them. The number of threads can cause inconsistency, creating more threads than needed an cause the application to never stop. Beside that, it can suffer the same problem as the DIANA, by trying to access arrays which sizes depend on the number of threads. A possible explanation for the variable "*out_filename*" causing crash is the fact that when the application tries to open an invalid file, it returns the value "-1", and this could be interpreted by CAROL-FI as a crash. Finally, if we change the clusters centers that are used in the recalculations, it can cause calculations that keep trying to find something that isn't there anymore, causing an infinite loop.

Duplication With Comparison (DWC)

We opted to use the temporal approach for the duplication technique since we had some problems using threads for the spatial approach in carol-fi. The overhead for the temporal duplication was obviously higher than the spatial as we do sequential calls to the kernel. In fact, the execution time was slightly more than two times higher than the one without fault tolerance techniques applied. This is expected, since in addition to the duplication, there is also the time cost of our SDC detection method. For the *k-means* algorithm, the temporal alternative can also be justified by the fact that it uses OMP Threads - we are using the power given by parallel threads to decrease the application run time, so it becomes more acceptable to explore the temporal duplication.

For each clustering algorithm, we implemented a results comparison - to determine if the outputs of the duplication were consistent with each other. Then, we opted to create an infinite loop as soon as we had updated the detections counting file to generate an apparent hang for carol-fi. This would make it easier for us to calculate the detected SDCs rate after the injections; If we just printed one of the two different outputs, it is unknown if carol-fi would treat that as an SDC or a masked injection (it would have a 50% chance of each one occurring). This also represents a practical approach, because in a real application, when an unrecoverable error is detected, we want to just stop the application or flag it as plainly wrong, because the results are no longer reliable.

As it was mentioned, each algorithm had its own technique for detecting differences between kernel executions. For AGNES, the method applied was basically comparing the two textual outputs character by character, and when some difference was found a flag was set, and a counter of detections in a file was incremented. As for DIANA, we opted for a more interpretative comparison - we compared each member of the dendrogram structures in memory. Finally, for *k-means*, for every cluster center coordinates of an execution, we checked if it was equal to the one at the same position for the other execution.

Fault Injection Results - DWC

Comparing the fault injection results between executions with or without DWC hardening, we can observe wildly different ratios between DUEs and SDCs. As expected, when using our implementation of the duplication with comparison, much more DUEs are observed, in exchange of SDCs. This occurs because detected SDCs are converted to hangs, with the use of infinite loops upon detection. The table below shows how many times the applications vulnerability factors changed from the original version to the hardened one, based on the data represented in Figure 7 - the PVFs with DWC.

Although it may seem that AGNES has an absurd increase in DUEs (even considering the huge reduction in SDCs), it actually isn't really problematic if we bear in mind that this algorithm had the lowest DUE-PVF of the three. With DWC, this difference gets neutered and the DUE-PVFs approach an equality. This peculiarity can be observed on Figure 6 (which shows the percentage of SDCs, DUEs and Masked Faults of the injections with DWC) where we can actually see, in comparison to the values of Figure 2 (the same values but with no hardening) a kind of "normalization of behavior" between the algorithms. Better said, the differences in the rates of DUEs between the clustering techniques are much less pronounced with DWC, and the SDC-PVF of AGNES gets comparable to the others - instead of being a huge spike, like wheat was seen with the unhardened version.

	Kmeans	**AGNES**	**DIANA**
Reduction in SDC-PVF	≈ 3.3 times	≈ 12.2 times	≈ 5 times
Increase in DUE-PVF	No change	≈ 9.7 times	≈ 1.5 times

Table 3 - Reductions in SDC-PVF and increases in DUE-PVF, using DWC

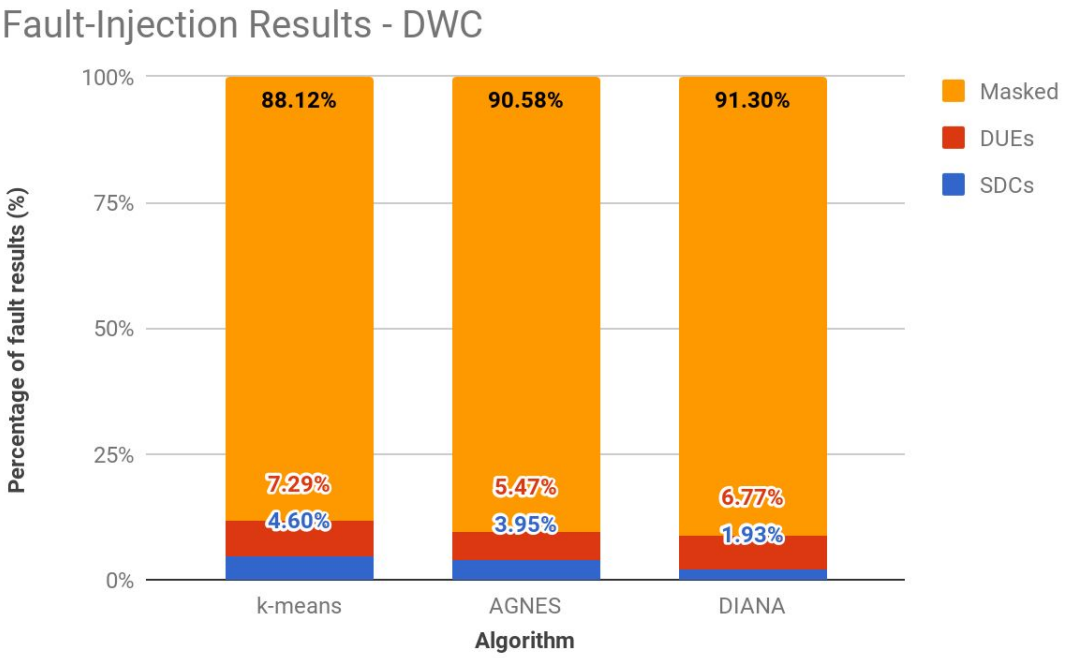

Figure 6 - The percentages of SDCs, DUEs and masked faults for each algorithm, using DWC

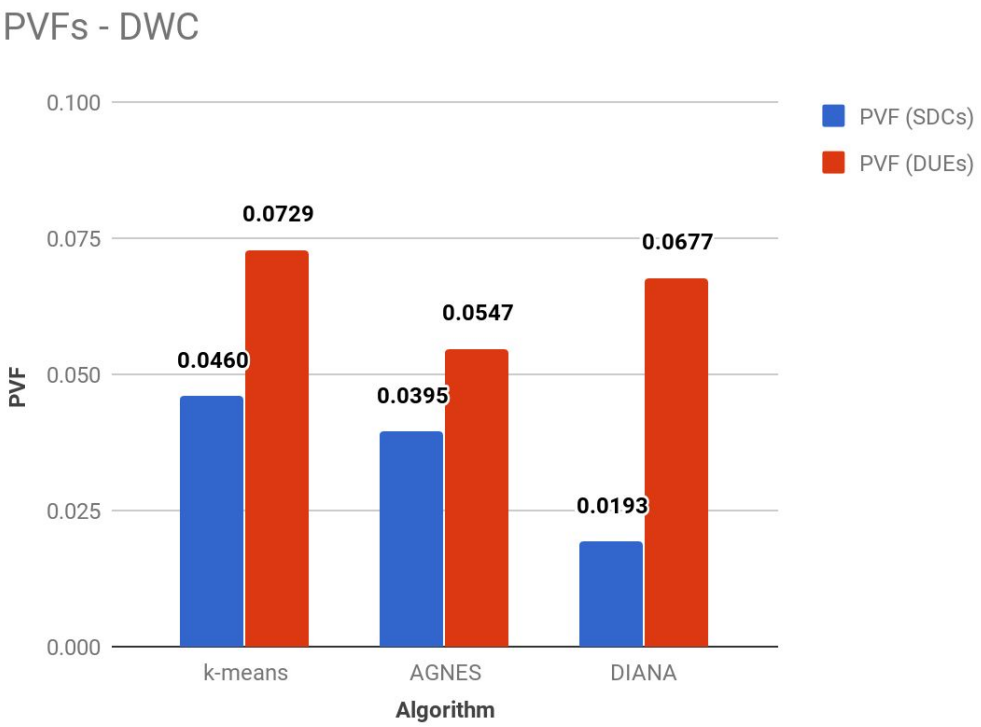

Figure 7 - The estimated values of PVF for each clustering technique, using DWC

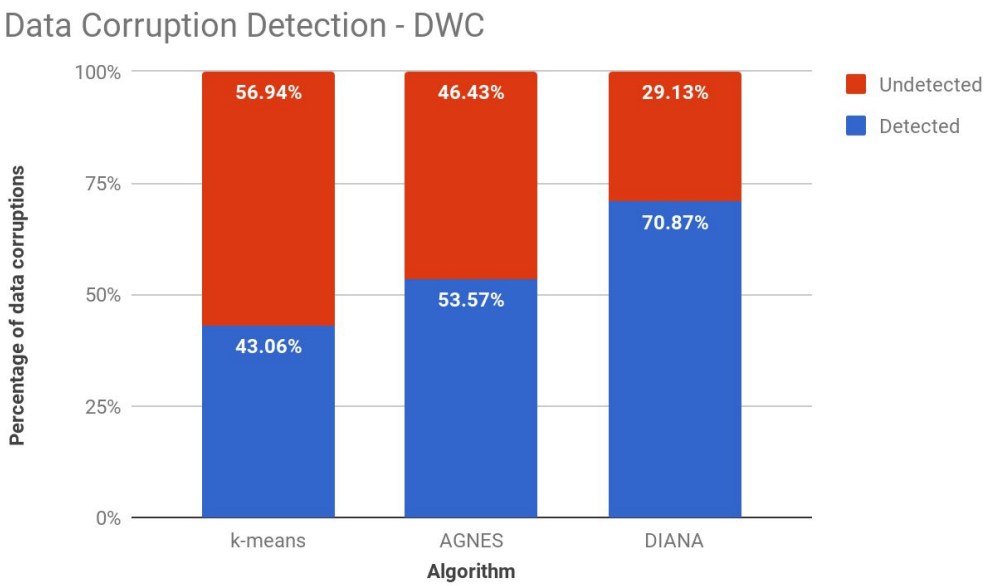

Figure 8 - Percentage of detected and undetected data corruptions, using DWC

From the chart of Figure 8, that shows the percentage of detected SDCs with DWC, we observe an amount of detections substantially lower lower than the expected for *k-means* and AGNES. We believe that this could be caused by fault injections affecting our input structures. Since we reuse the read input in the second kernel execution, if a fault is injected before the two kernels run, both of them would do a correct (and equal) computation, but over a wrong input data, causing an SDC (garbage in, garbage out).

Time Window Analysis - DWC

Figure 9 - The percentage of SDCs produced in each time window, using DWC

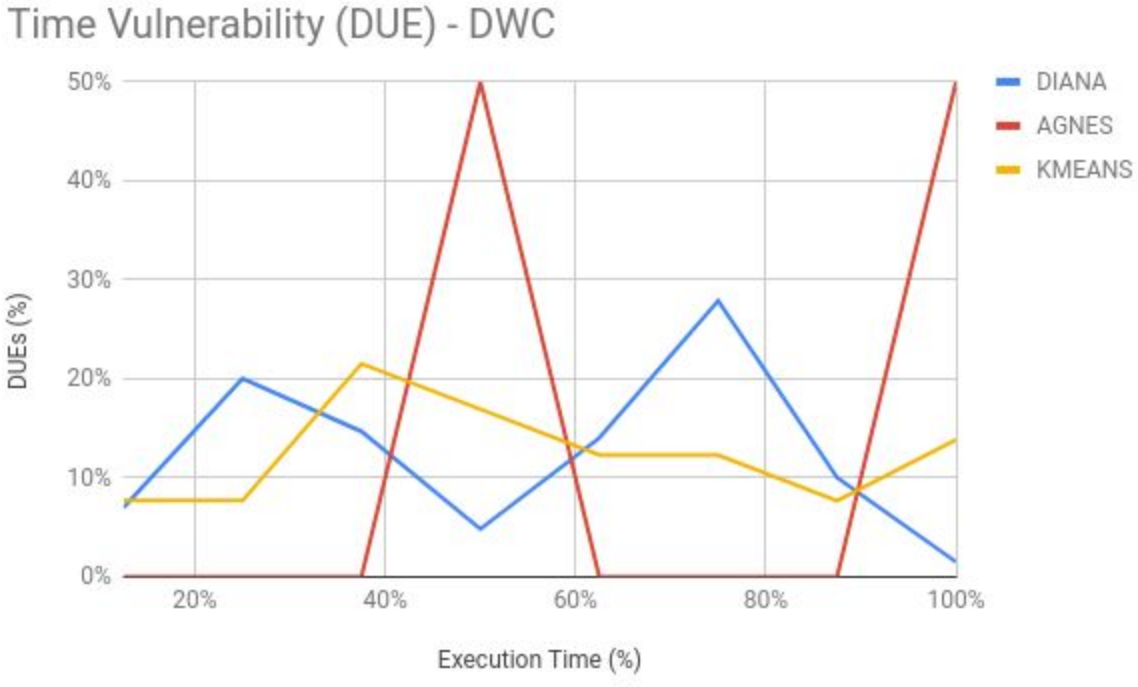

Figure 10 - The percentage of DUEs produced in each time window, using DWC

The SDC time vulnerability of the three algorithms with DWC, plotted in Figure 9, reflects the characteristics of the said hardening technique. There is a visible increase in the vulnerability after the 85% mark of the execution time, which is approximately when the comparison of outputs starts. That said, we believe that faults affecting variables or memory regions that have already been compared and verified to not have data corruption are the ones to blame, since the comparison would naturally never detect them. There is also a high-vulnerability section of the graph by the beginning of the executions. This indicates that the inputs may be corrupted before they are passed to the algorithms' kernels, producing two equal outputs (therefore not flagged by our detection), but both wrong.

In AGNES, we can observe a very similar behavior when compared to the time analysis in the unhardened version. The peaks are located in the beginning and in the end of the curve as in the first scenario. That said, we can assume that even with full duplication this two regions of the code are highly sensitives to fault injections. On the other hand, analysing the injections in the middle of the execution we can assume that this region became the very opposite of highly sensitive to fault injections. There was almost an absence of SDCs in this area of the code when the code was full duplicated. This represents that any fault injected in that time can be easily masked or detected when using this fault tolerance technique. In the DUE behavior graph, on the other hand, the data actually confirms what was expected since we can see that the peak of DUEs occurs while the code's execution is generating the cluster hierarchy.

For the *k-means* curve, we notice that the peaks seen on the previously graph are gone, making the hardened curve more evenly spread than the unhardened one. The DUE analysis shows about the same curve from before, but now we can see an increase in the DUE percentage, mainly due to the forced hang state that we send the application with our handmade SDC detection.

Critical Variables - DWC

Variables Sorted By SDC-PVF

AGNES			
PVF	**#SDCs**	**Variable**	**Type**
33.33	1	loc	?
20.00	2	prodp	?
6.98	3	_IO_file_jumps_maybe_mmap	?
6.56	4	__GI__IO_file_jumps	?
4.65	2	_IO_file_jumps	?

Table 4.1 - The most critical variables for SDCs in AGNES, using DWC

DIANA			
PVF	**#SDCs**	**Variable**	**Type**
100.00	2	__x	?
100.00	1	exponent	?
75.00	12	__x	?
75.00	6	__k	?
64.00	16	cluster_ptr	struct *

Table 4.2 - The most critical variables for SDCs in DIANA, using DWC

k-means			
PVF	**#SDCs**	**Variable**	**Type**
100.00	2	nptr	?
100.00	1	pt1	float *
50.00	1	min_dist	float
41.67	5	nclusters	int
40.00	6	numAttributes	int

Table 4.3 - The most critical variables for SDCs in k-means, using DWC

Observing the Table 4.1, we see that now the most critical variables for AGNES are actually related to file operations detected by the gdb. We can correlate that to the high vulnerability presented by this version of the algorithm by the end of its execution (see Figure 9), when it writes its output to a file.

In Table 4.2, which relates to DIANA with DWC, three of the most critical variables for SDCs (__*x*, __*x* and __*k*) are from the C++ standard Map (std::map) and tree implementations, which is consistent with our theory about the vulnerability to faults in variables that have already been compared, mentioned in the DWC time window analysis section. The same principle most likely applies to faults in the variable *cluster_ptr*, the last one of the list, which is a linked-list pointer inside one of the dendrograms. can also be The compared outputs are dendrograms, which have two main structures stored in maps, so it makes sense for the variables in the list to be critical.

Table 4.3 shows us almost the same variables from the previous one (for the unhardened variables). We know that the faults are injected randomly, but the most used variables are more likely to receive an injection. We are duplicating the clustering operation, that is the mainly operation for the application, therefore, they are going to continue as critical variables, but the SDCs would be divided between the two different instances of the same variable.

Variables Sorted By DUE-PVF

AGNES			
PVF	**#SDCs**	**Variable**	**Type**
100.00	11	op1_word	?
66.67	2	wide	?
22.22	2	vp	?
4.65	2	_IO_file_jumps_maybe_mmap	?
2.33	1	_IO_file_jumps	?

Table 5.1 - The most critical variables for DUEs in AGNES, using DWC

DIANA			
PVF	**#SDCs**	**Variable**	**Type**
100.00	1	point	int
54.00	54	cluster_to_divide_index	int
51.61	16	loop	int
50.00	1	points_2_size	int
44.17	53	n_features	int

Table 5.2 - The most critical variables for DUEs in DIANA, using DWC

k-means			
PVF	**#SDCs**	**Variable**	**Type**
61.54	8	nloops	int
50.00	1	tid	int
38.46	5	nthreads	int
30.77	4	num_omp_threads	int
30.00	3	new_centers	float **

Table 5.3 - The most critical variables for DUEs in k-means, using DWC

When analysing Table 5.1, one can make the same already mentioned considerations about AGNES vulnerability rise by the end of the execution being caused by the file operations: some variables of the list are related to it (the last two). The other variables are related to comparisons and printing to the standard input device - operations inherent of the final comparison of the produced outputs and reporting the result of the execution.

As for Table 5.2, some of the variables are unchanged, comparing this table to Table 2.1 (DUE-critical variables for unhardened DIANA), and some variables from the SDC-critical list of unhardened DIANA, in Table 1.2, show up as well. This is expected, since we are converting the final effect of some faults that would otherwise cause an SDC to a DUE, by doing duplication with comparison. Better said, some faults that would corrupt the output of the program are now detected, and when this happens, the program hangs, causing a DUE. For *k-means*, in Table 5.3, in we can observe the same behavior as DIANA, since they share some parts of the code and they have been implemented and hardened using the same concepts.

Selective Hardening (SH)

When implementing selective hardening of the *k-means*, DIANA and AGNES, we considered the lists of most critical variables to SDC/DUE PVF of the three algorithms (from Table 1.1 to Table 2.2). All of the variables were duplicated, memory and operation-wise: allocating heap or stack memory anew, and repeating the original operations to the duplicated versions of the variables.

This duplication is accompanied by a series of comparisons, so that every time one of the variables' value has to be used in an operation, the two duplicates are checked to see if their values are consistent. If so, the program can proceed as it normally would, with the just mentioned instruction using one of the equally valid variables. If it happens that the two duplicates are not equal, then a data corruption has been detected - the program increments the detection count in a file and proceeds to loop indefinitely, waiting for the fault injector to stop its execution.

Fault Injection Results - SH

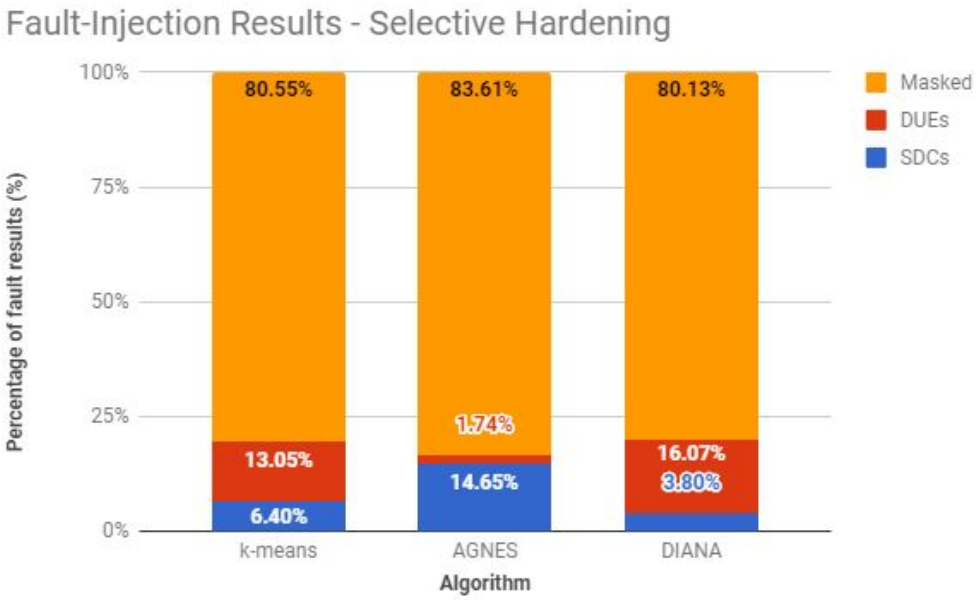

Figure 11 - The percentages of SDCs, DUEs and masked faults for each algorithm, using SH

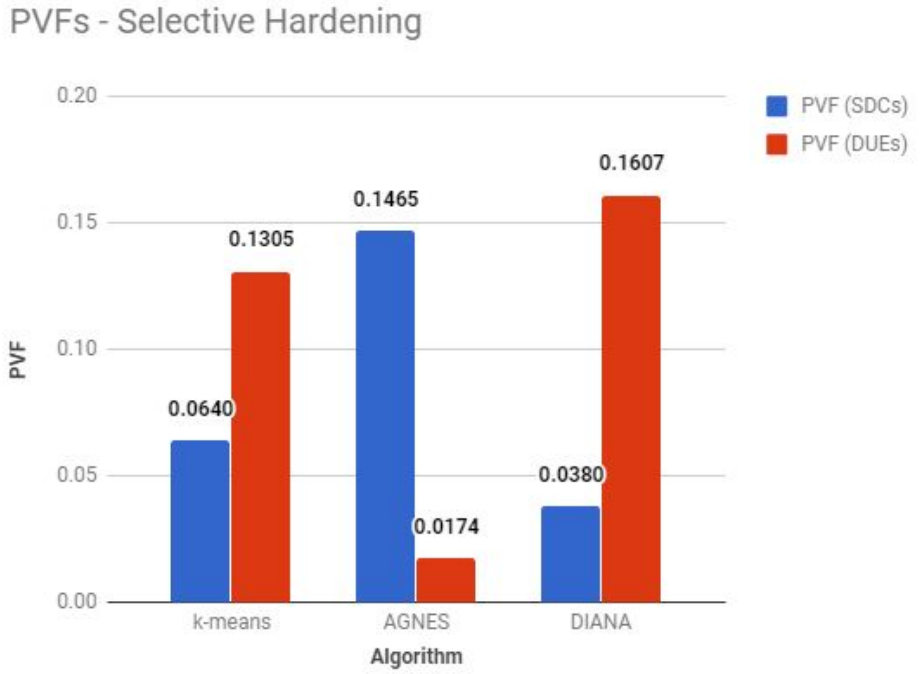

Figure 12 - The estimated values of PVF for each clustering technique, using SH

The results of the fault injection on the codes with selective hardening are shown in Figure 11, while the consequent PVFs, for SDCs and DUEs, are illustrated in Figure 12. To better grasp the changes in this quantities, we elaborated Table 6, which shows by how many times the SDC-PVF decreased and the DUE-PVF increased in comparison with the unhardened results. We see less drastic values in this table, compared to the ones in Table 3 (related to DWC results). This shows that this hardening technique was not so effective as the full duplication, but it is for sure more efficient, since the execution times are way shorter than running the kernels two times in a row, being practically the same as the unhardened version.

	Kmeans	**AGNES**	**DIANA**
Reduction in SDC-PVF	≈ 2.3 times	≈ 3.2 times	≈ 2.5 times
Increase in DUE-PVF	≈ 1.8 times	≈ 3.1 times	≈ 3.6 times

Table 6 - Reductions in SDC-PVF and increases in DUE-PVF, using selective hardening

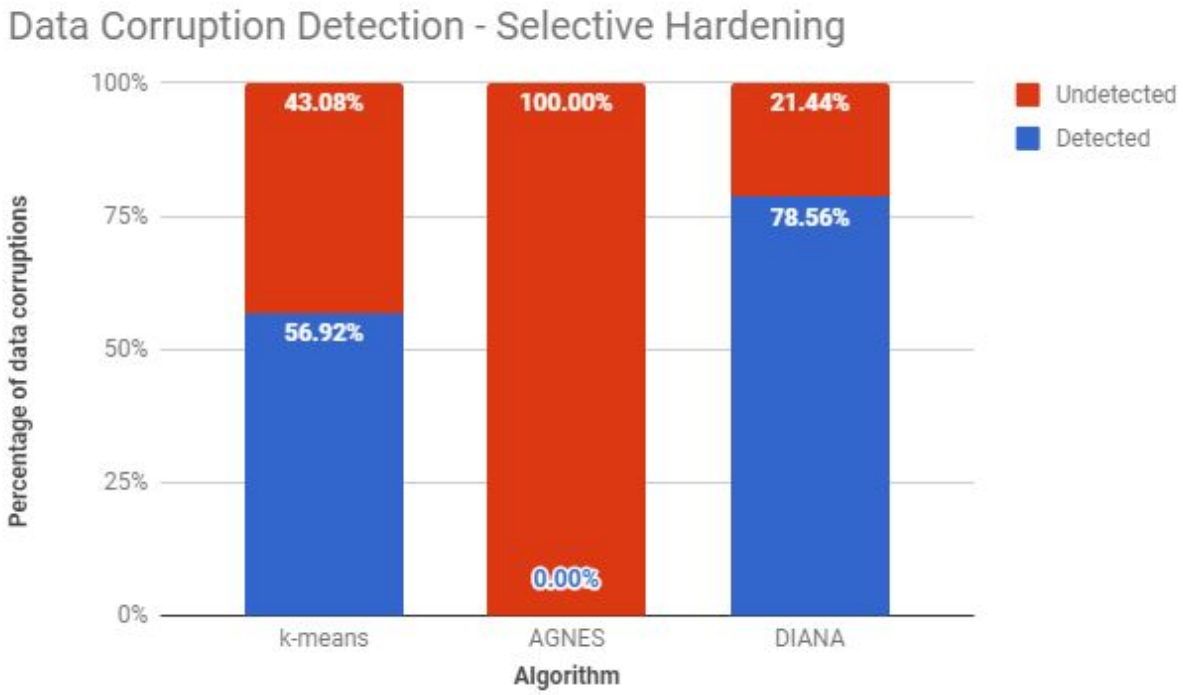

Figure 13 - Percentage of detected and undetected data corruptions, using SH

Both *k-means* and DIANA had a raise in detection rates, when using selective hardening, compared to their DWC versions. At first, this may seem strange, since duplicating and comparing the whole kernel (what is done when using DWC) should, in theory, yield a higher or equal rate of detections than simply duplicating and comparing some variables. That said, we think that what occurs is that the overhead of comparing two entire dendrograms is so high that faults that affect already compared portions of them contribute majorly to the rate of undetected data corruptions, resulting in a higher detection rate for selective hardening when compared to DWC, since SH doesn't have that same overhead. This theory gains strength when we consider the differences between Figure 9 and Figure 14: the huge spike of vulnerability by the end of the execution of DWC gets squashed down when using SH, indicating that these types of faults just mentioned are occuring way less.

An anomaly occurred in AGNES' fault injection as we can see from the graph above. We previously commented that the critical variables for this application were the ones found in external C codes from operational libraries found by gdb. Since we did not have data from the injections in our source code due this fact our selective hardening was useless in this case. It is important to mention that our selective hardening method was perfectly working when we manually injected faults in the critical variables. This strange behavior was present only when we started our execution with carol-fi fault injector. Despite this unfortunate incident we still can see a normal behavior in the PVF data for SDC and DUE according to Table 6.

Time Window Analysis - SH

Figure 14 - The percentage of SDCs produced in each time window, using SH

Figure 15 - The percentage of DUEs produced in each time window, using SH

As it was previously explained in the section "Fault Injection Results" of Selective Hardening, Figure 14 shows a difference in magnitude of the SDC vulnerability in the end of the executions of the clustering techniques compared to Figure 9 (see Time Window Analysis of the unhardened code). This occurs mainly because of the final kernel outputs comparison being eliminated with selective hardening, opting for comparing variables throughout the algorithms' executions.

DIANA, in Figure 14 and Figure 15, shows peaks of vulnerability similar to the ones found on the unhardened version of the code, by the 65% of the execution time mark. We believe this occurs for the same reasons stated in the explanations of Figure 9 and 10, under the time window analysis section for the original codes. The time window behavior of AGNES is also similar to the unhardened one, but the peaks of vulnerability in the middle of the execution get smoothed out, probably by the selective hardening. The code also becomes more vulnerable to DUEs in this sections, which makes sense, since we are trading data corruption for hangs.

Critical Variables - SH

Variables Sorted By SDC-PVF

AGNES			
PVF	**#SDCs**	**Variable**	**Type**
100.00	22	op1_word	?
96.00	24	__free_hook	?
93.75	15	__morecore	?
86.36	19	__after_morecore_hook	?
73.68	14	__malloc_hook	?

Table 7.1 - The most critical variables for SDCs in AGNES, using SH

DIANA			
PVF	**#SDCs**	**Variable**	**Type**
100.00	2	list_of_points	int *
50.00	5	n_features	int
50.00	1	t	?
50.00	1	points_1	int **
42.22	19	new_centers_len	int *

Table 7.2 - The most critical variables for SDCs in DIANA, using SH

k-means			
PVF	**#SDCs**	**Variable**	**Type**
60.71	17	nptr	?
45.95	68	j	int
42.86	9	nthreads	int
42.86	3	new_centers_len	int *
40.63	13	pt	float **

Table 7.3 - The most critical variables for SDCs in k-means, using SH

The tables above (7.1 to 7.3) show variables that were not in the original lists of SDC-critical variables (tables 1.1 to 1.3). This is expected, since we are hardening the ones from that anterior lists, making the effects of other ones come to surface. That said, we still see some variables that were on the original lists - which is explained by the fact that the hardened variables could be corrupted before duplication, making its copied value equally wrong.

Variables Sorted By DUE-PVF

AGNES			
PVF	**#SDCs**	**Variable**	**Type**
60.00	18	global_max_fast	?
25.00	2	user_entry	?
20.00	1	_dl_argv	?
4.00	1	thread_arena	?

Table 8.1 - The most critical variables for DUEs in AGNES, using SH

DIANA			
PVF	**#SDCs**	**Variable**	**Type**
100.00	298	n_features_2	int
100.00	289	n_features_1	int
100.00	287	n_clusters_in_anterior_level_1	int
100.00	261	n_clusters_in_anterior_level_2	int
100.00	240	cluster_index_2	int

Table 8.2 - The most critical variables for DUEs in DIANA, using SH

k-means			
PVF	**#SDCs**	**Variable**	**Type**
100.00	160	nclusters_duplicated	int
100.00	155	nclusters	int
98.29	172	numAttributes_duplicated	int
96.36	159	numAttributes	int
81.82	18	loop	int

Table 8.3 - The most critical variables for DUEs in k-means, using SH

Most of the variables that appear in Table 8.2 and Table 8.3, being critical for DUEs in *k-means* and DIANA with selective hardening, are variables that are duplicated by this technique. Most of them are the same ones from Table 1.2 and 1.3, the lists of the most critical variables for SDCs in the original codes.

Since selective hardening on AGNES could not detect any data corruption, we don't see the same behavior as the one just mentioned for the other clustering techniques. After all, no hangs are being generated from the "hardened" variables, and the original DUE-PVF of this algorithm was already really low compared to the others. It is also relevant to mention that the majority of fault injections that caused an SDC/DUE in AGNES were applied in comparison variables such as as cmp.c source code. This data is evidence that the algorithm has a high probability of having corrupted output values due to faults injected while comparison was being executed.

Hardening Techniques Comparison

PVF Comparison

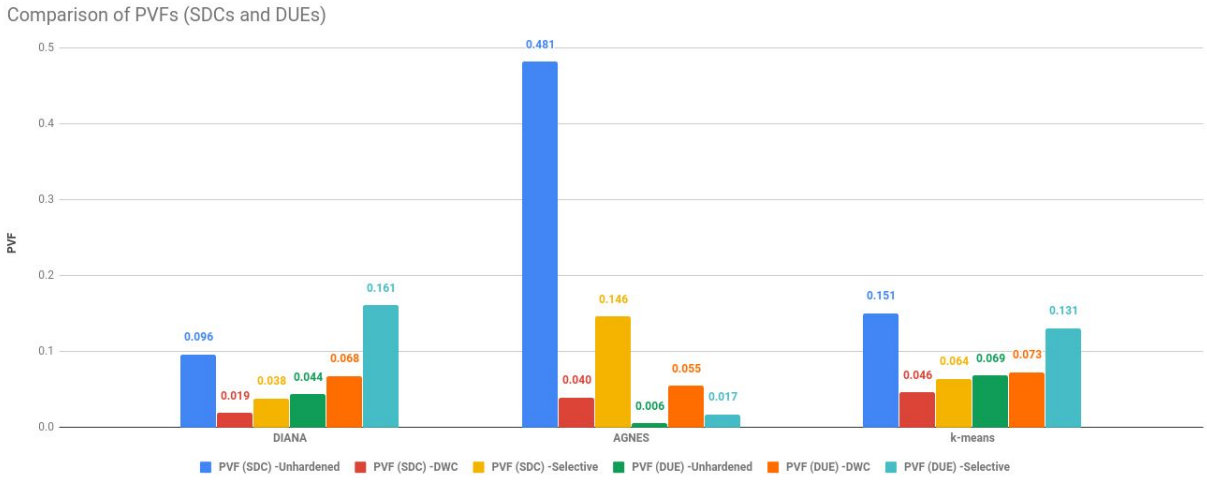

Figure 16 - SDC/DUE PVFs of all algorithms, with unhardened code, DWC or SH

By compiling the results of the 9 fault injections (3 algorithms with 3 different hardening levels), we can plot a graph of the PVFs of each version of each clustering technique, for DUEs and SDCs. That is Figure 16, a huge comparison that puts our results into perspective.

Many comparisons of the results were presented throughout this report, and they can be confirmed by the analysis of this chart. Some considerations worth noting are, for example, the huge difference in SDC-PVF from AGNES to the other two algorithms in their unhardened versions. There is also a really similar overall behavior between DIANA and *k-means*, justified by their similarities in implementation (one uses the other's kernel), with the numbers for DIANA being lower maybe due to the added complexity that could mask some of the faults.

Comparing DWC and SH results is made easy with this graph. We can see that all the SDC-PVFs of selective hardening are higher than the corresponding DWC ones (compare yellow and red bars to see that difference). DIANA and *k-means* show higher DUE-PVFs using SH compared to using DWC, but AGNES has the opposite behavior - though the three clustering algorithms show higher DUE-PVF when using any of the two hardening techniques, which is expected since the detections cause hangs.

Data Corruption Detection Comparison

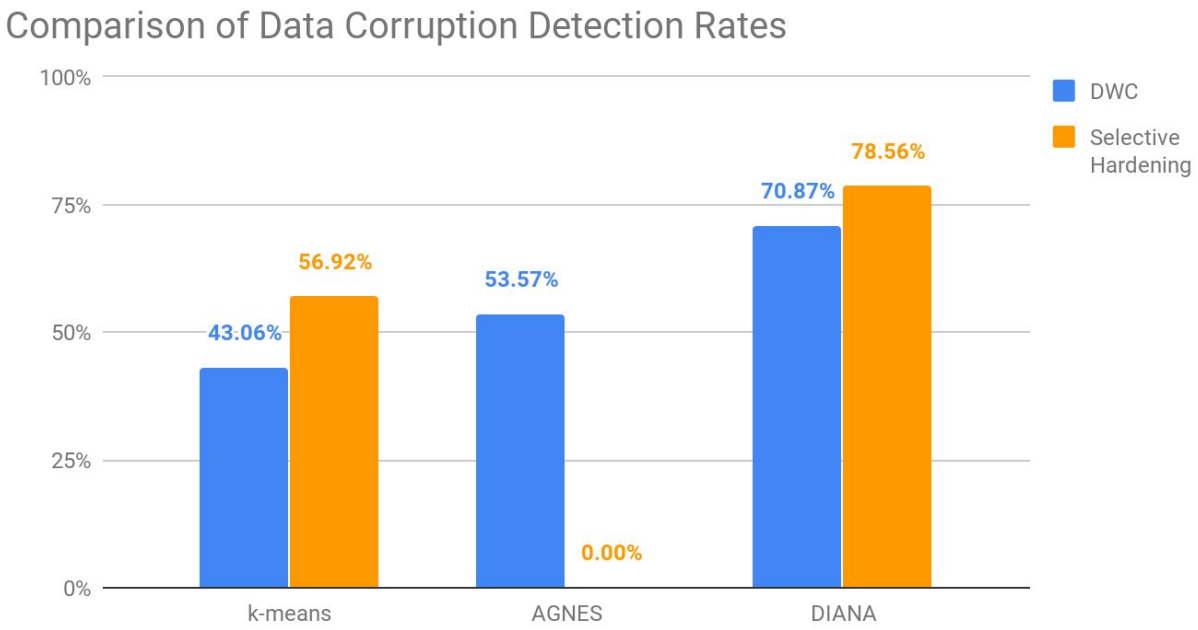

Figure 17 - Data corruption rates with DWC and SH, for the three algorithms

As it was explained before in this document, selective hardening yielded higher detection rates because of the faults that occur in DWC during the final (and lengthy) comparison, but in memory regions of the outputs that have already been compared. We believe that if the comparison times were shorter, the DWC results could have been more adequate, but with the version that was implemented, we judge the selective hardening approach both more efficient and more effective.

Finally, it can be said that DIANA is clearly the best candidate to be chosen as a fault tolerant clustering algorithm, since both DWC and SH detection rates were between 70% and 80%, and the PVFs were the lowest overall (as it can be seen on FIgure 16). That said, we do think that the implementations of the hardening techniques on the other two algorithms could be further tested by other academics, to study the effects of slightly different implementations of DWC and SH (such as changing the comparison modes, or the output formats).

References

[1] D. Oliveira, L. Pilla, N. DeBardeleben, S. Blanchard, H. Quinn et al. 2017. Experimental and Analytical Study of Xeon Phi Reliability. ACM International Conference for High Performance Computing, Networking, Storage and Analysis, SUPERCOMPUTING. Denver, CO, USA. http://www.lume.ufrgs.br/handle/10183/171994 .

[2] V. Sridharan, D. R. Kael. 2010. Using PVF Traces to Accelerate AVF Modeling. Boston, MA, United States. Northeastern University. https://www.researchgate.net/profile/David_Kaeli2/publication/228960028_Using_PVF_Traces_to_Accelerate_AVF_Modeling/links/0deec5226219219ea4000000/Using-PVF-Traces-to-Accelerate-AVF-Modeling.pdf .