Determining Acceptance Tests For Application-Level Fault Detection

Eric Ciocca, Israel Koren, C. Mani Krishna

Abstract— **Faults are often difficult to detect, especially when the detection algorithm does not understand the semantic context of the faulty data. By allowing the application itself to take on the responsibility of detecting faults, significant savings can be made in incurred error penalty. However, such an approach to fault detection is inherently bound to be inaccurate sometimes. Inaccuracies may be the result of either faulty elements which were not detected, or nonfaulty elements which were wrongly considered to be faulty. The end user must consider not only the error penalty of missed faults, but also the extra runtime which may come with a false alarm.**

Our approach to software level fault tolerance allows for levels of customization within each acceptance test. As such, the correct configuration of these error bounds is a matter left to the end user. The end user must also determine a balance among the acceptance tests, such that the serialization of multiple tests results in not only the best fault detection, but also a low overhead. We demonstrate these principles as applied to NASA's Orbital Thermal Imaging Spectrometer (OTIS) application.

I. Introduction

Faults that occur in a system may manifest themselves as various types of errors. Faults may be destructive and obvious enough to disable entire nodes and render the hardware useless, or may be as localized as to only flip a single bit in the memory. Faults of the latter type are very difficult to detect, yet the consequences may be more severe than those of a hardware fault. While hardware-disabling faults can be instantly detected by any number of watchdog devices, faulty data may slip by undetected and pose as valid data. The end user, trusting that this data is fault-free, will go on to use it, never suspecting that it is in fact erroneous.

To prevent this, robust systems may incorporate measures of fault detection. This fault detection may come via custom hardware or low-level modifications, but will most likely involve some sort of redundancy. This redundancy may be as simple as the information redundancy of a parity bit, or it may be more complicated, such as the use of multiple versions of software to validate results. Not all techniques can be applied to all applications, so the entire system and the operating environment must be considered before determining the best option. It may help to use application-specific approaches to fault detection.

Using application-specific characteristics for fault detection is a straightforward way of weeding out erroneous data. An application (or the developer of an application) knows the format, range, and other subtleties of input, output, and intermediate data. By knowing what is expected, a program can rule out unexpected data as the product of a fault. Such an acceptance test can be as simple as a logical sanity check, or as complicated as a transformation of the data to test its feasibility in another domain. By making the acceptance test as simple as possible, the overall cost for fault detection can be lessened. While this may not serve as a completely accurate fault detector, it can be used to supplement and reduce the overhead of other fault detection schemes.

This research was supported in part by NASA, JPL contract number 1223474, and by NSF grant number CCR-0104482.

II. Description of Problem

The backbone of application-level fault detection relies on having regular and predictable data characteristics. Finding the domains in which data is predictable requires knowledge of what the input and output data represent, and the characteristics inherent to that set. If, for example, we are considering a set of input and output that deals solely with natural phenomena, a series of restrictive rules can be created based on the environment of the experiment and the physical laws associated with the readings. Any test that holds data to any criteria to determine its validity is in this context called a fault "filter."

Often when attempting to detect faults, there arises a tradeoff between detection accuracy of faults, and detection mistakes via genuinely nonfaulty data. By creating filters which are too lax, faulty data will slip "under the wire," and not be detected. By creating filters which are too strict, many nonfaulty data elements will be incorrectly identified. The tradeoff between these may manifest itself as error (missed faults) versus extra runtime (false alarms), but this may change from application to application. Ultimately, the tradeoff must be balanced by the end user.

Unfortunately, finding the optimal calibration for any filter or set of filters is often too complex to solve numerically, even if the tradeoffs can be expressed in a formulaic fashion. Another method for calibrating the filters relies on trial and error, refining and retrying different values until the best ones are found. The presence of multiple filters further complicates this process, making the discovery of optimal filters akin to shooting in the dark. The method proposed in this paper is a form of intelligent trial and error, which attempts to use observed trends in data to reduce the areas where trials are required.

The application considered in this paper for fault detection makes use of the natural characteristics of temperature. OTIS, the Orbital Thermal Imaging Spectrometer, is an application run in orbiting satellites to collect thermal information from planets. OTIS is part of NASA's REE application group [1], a collection of applications intended

to run in orbital, spacebound, and extraterrestrial environments.

Temperature has two immediately straightforward characteristics:

• Natural Bounds - The data represents a natural phenomenon within a small region, so it will fall within a predictable range. For example, while doing a temperature survey of the Earth, we should not expect to find anything too far below freezing, nor exceeding the boiling point of water (with the exception of "hot spots" like volcanoes). When surveying a localized geographic area of only a few meters or kilometers, as a satellite would do, the cutoff values can be placed even tighter. Bounds could be described by the target area, such as "tropical" or "arctic" bounds.

• Spatial Locality - The data within a small geographic area will not only fall within a certain range, but also change gradually. While dramatic sudden changes (hot or cold spots), are not unheard of, heat dispersion tends to average the spots into their surrounding area. Even so, the temperature within the spot is usually locally consistent, so the only inconsistent data will be along the spot's edges.

By leveraging on these characteristics, we hope to be able to rule out any data which is altered by faults. The exact method of discovering the characteristics of particularly adjusted filters, as well as the associated trade offs, will be discussed below.

III. Related Work

Application-specific fault detection is a prerequisite for our approach to Application-Level Fault Tolerance (and Detection), ALFTD [2]. This scheme is a general approach to application modification that allows for an application to also compensate for faults it has detected. The method of tolerance varies from application to application, but OTIS and previous applications have accomplished this through scaled software redundancy. In this, processors are responsible not only for their own work (at full resolution), but also a redundant scaled copy of a neighboring processor. Scaling may be performed by reducing the input dataset, the accuracy of functions, or in any other way restricting the amount of work done while still producing acceptable results.

Further advantage comes when considering applications where it is possible to make the redundant copies optional. If faults are statistically infrequent, using a method which is reactive to fault detection may have a lower overhead than proactive application or algorithmic [3] fault tolerance. If possible, we would also like to single out particular segments of data that are potentially faulty and target them for redundant calculation. At the same time, we would also like to concretely identify as many nonfaulty data segments, so as not to do unnecessary redundant calculations on them.

Application-level fault detection allows for implementation of application-level fault tolerance without reliance on system-level fault detection hardware. It also allows for redundant verification through processes within a single software, instead of relying on multiple software versions [4]. Previous forms of Application-Level Fault Tolerance [5] only compensated for faults which were destructive enough to prevent processors from producing output. However, in particular environments this may be insufficient. The nature of space-bound systems exposes them to a greater number of charged particles, which may manifest themselves as very discrete faults [6]. Inclusion of application-level, semantically guided fault detection is intended to supplement existing system and hardware level fault tolerance, and compensate in cases where lower-level fault tolerance breaks down. The application semantic basis for ALFTD's fault detection enables it to pick up on faulty hardware passing off erroneous data as valid, as well as data which has become corrupt by other means. Actually tolerating these types of failures requires a redundant source to provide the valid data, but in cases where redundancy isn't possible it is still possible to at least signal the presence of a suspected fault.

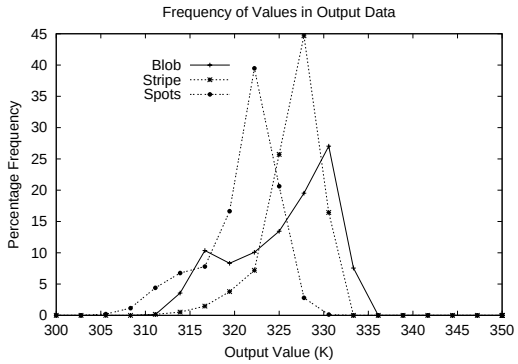
IV. Approach

Our goal is to find the cutoff values beyond which data is most likely faulty. To begin with, the data itself must be investigated in order to find the typical values of nonfaulty data. By finding the characteristics of nonfaulty data, it can be decided which data ranges should be "inside" the filter boundaries. OTIS filters utilize two data domains, the temperature (measured in Kelvin) and "spatial locality" - measured as the difference in temperature between adjacent pixels. The ranges of three different datasets (named Blob, Spots, and Stripe) are illustrated as a frequency plot in Figure 1.

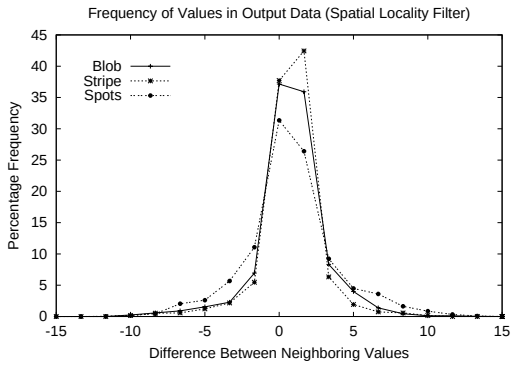
These plots show that the data falls within a fairly narrow data range. Being able to detect faults is now a matter of detecting any data which falls outside these narrow ranges. The potency of this approach, however, depends on the type and intensity of faults which are to be detected.

If, for example, faults manifest themselves by resetting portions of memory to zero, or to gibberish values, the chances of the faulty data falling inside the relatively slim filter range is negligible. If faults are occurring because of charged particles flipping single bits in memory, the error may manifest as a small offset in value. In some cases the error may be small enough to be negligible (such as hitting the least significant bits), or may be large enough to be immediately apparent (increasing the values by orders of magnitude). We concern ourselves with the intermediate intensities of faults, those which create some noticeable - yet not outrageous - offset to existing values. With an offset range of plus or minus 10%-30%, it is possible for values to both fall outside the existing acceptable data range, or to still remain "inside." While errors themselves are statistically infrequent, and the chances of an intermediate error such as this even more so, this is an extreme fault case used to exercise the worst-cast scenario for application-level fault detection.

As there is the chance for data with errors of this intensity to be even within the typical range of fault-free data, a strict (numerically narrow) filter would result in

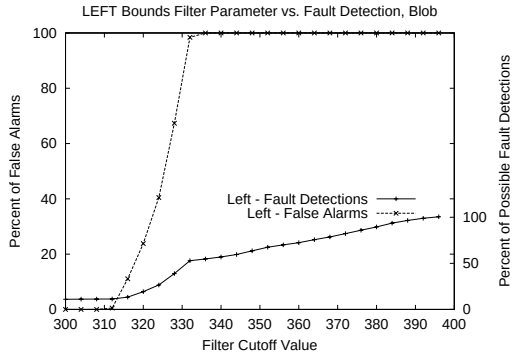


(a) Data Value Frequency

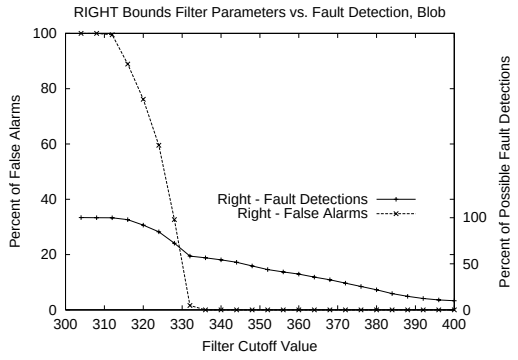


(b) Data Locality Frequency

Fig. 1. Frequency Plots of Three Sample OTIS Outputs



(a) Bounds Filter (Left Cutoff)



(b) Bounds Filter (Right Cutoff)

Fig. 2. Various Side-Specific Filter Values for the Bounds Filter

the most faults detected. But a tradeoff must be made between faulty data being detected, and nonfaulty data "false alarming" the filters. To study this case, a set of results was generated using moderately faulty input data (adding or subtracting 10%-30% to the existing values). The output is in a 50×50 array of floating point values. Each "pixel" had approximately a 25% chance of being erroneous, and the intensity of each fault was randomly determined. The results discussed in this article are the averaged values over ten such experimental repetitions.

Each filter has two boundary values, the lower (left) bounds and the upper (right) bound. Data outside these bounds is suspected to be faulty. Each boundary value can be set independently. Different settings for each will incur different amounts of successful fault detections and false alarms from the fault-injected data. The effect of each filter bound is cumulative - the total number of faults detected in a system is the sum of faults detected by the lower and upper bounds. Likewise, the total number of false alarms is the sum of the false alarms incurred by the upper and lower bounds. The effects of the lower and upper bound can be seen in Figures 2a and 2b, respectively.

There are approximately 700 randomly placed faults to be detected in each dataset, and 2500 pixels in the entire field. The left y-axis ticks give the percent of false alarms out of the maximum number of false alarms. The maximum number of false alarms in this example is approximately 1800, the number of pixels without randomly placed faults in them. The right y-axis shows the fault hit tally as it is relative to the total number of pixels with randomly assigned faults. This allows us to see how the number of false alarms and fault hits scale for a particular cutoff, while still allowing us to determine at which points 100% fault detection is being achieved. Using a plot such as this, a cap could be placed on the number of fault misses or false alarms (per side), and the optimal placement of the left and right filter bounds can be determined.

By overlaying the graphs of the two side-specific filter boundaries, more general trends can be observed, as seen in Figure 3. Noting that the intersection of the "Hit Faults" graphs occurs to the right of the intersection of the "False Alarms" graph, the conclusion can be drawn that despite the uniform distribution of *faults*, the associated *errors* manifest themselves mostly by increasing the data values. This is a peculiar quality unique to OTIS, but demonstrates an important part of filter determination. If a system-wide fault detection rate of a particular accuracy is desired, we cannot assume that each side of the filter will be equally responsible for detecting those faults. With the particular type of fault in the given application, the right side of the

Complete Bounds Filter Fault Hits Cols = Left Side, Rows = Right Side									
	302	**304**	**306**	**308**	**310**	**312**	**314**	**316**	**318**
315	98.9%	98.9%	99.1%	99.1%	99.1%	99.1%	99.2%	-	-
317	96.0%	96.0%	96.1%	96.1%	96.1%	96.1%	96.3%	99.4%	-
319	92.7%	92.7%	92.9%	92.9%	92.9%	92.9%	93.0%	96.1%	98.0%
321	90.6%	90.6%	90.8%	90.8%	90.8%	90.8%	90.9%	94.0%	96.0%
323	88.1%	88.1%	88.3%	88.3%	88.3%	88.3%	88.4%	91.5%	93.5%
325	82.7%	82.7%	82.8%	82.8%	82.8%	82.8%	83.0%	86.1%	88.0%
327	77.9%	77.9%	78.1%	78.1%	78.1%	78.1%	78.2%	81.3%	83.3%
329	69.7%	69.7%	69.8%	69.8%	69.8%	69.8%	70.0%	73.1%	75.0%
331	62.0%	62.0%	62.1%	62.1%	62.1%	62.1%	62.3%	65.4%	67.3%
333	59.0%	59.0%	59.2%	59.2%	59.2%	59.2%	59.3%	62.4%	64.4%

TABLE I
PERCENT OF FAULT HITS BY LEFT AND RIGHT SIDE FILTERS

Complete Bounds Filter False Alarms Cols = Left Side, Rows = Right Side									
	302	**304**	**306**	**308**	**310**	**312**	**314**	**316**	**318**
315	92.6%	92.6%	92.6%	92.6%	92.6%	93.0%	96.5%	-	-
317	84.6%	84.6%	84.6%	84.6%	84.6%	85.0%	88.6%	95.7%	-
319	78.3%	78.3%	78.3%	78.3%	78.3%	78.7%	82.3%	89.4%	97.0%
321	72.0%	72.0%	72.0%	72.0%	72.0%	72.4%	75.9%	83.1%	90.6%
323	64.2%	64.2%	64.2%	64.2%	64.2%	64.6%	68.1%	75.3%	82.8%
325	53.7%	53.7%	53.7%	53.7%	53.7%	54.0%	57.6%	64.7%	72.3%
327	40.7%	40.7%	40.7%	40.7%	40.7%	41.0%	44.6%	51.7%	59.3%
329	24.2%	24.2%	24.2%	24.2%	24.2%	24.6%	28.1%	35.3%	42.8%
331	5.0%	5.0%	5.0%	5.0%	5.0%	5.4%	8.9%	16.1%	23.6%
333	0.1%	0.1%	0.1%	0.1%	0.1%	0.4%	4%	11.1%	18.7%

TABLE II
PERCENT OF FALSE ALARMS DETECTED BY LEFT AND RIGHT SIDE FILTERS

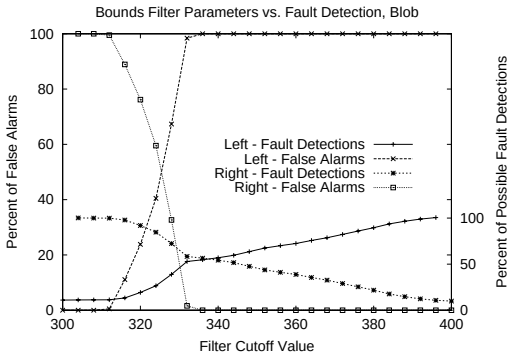


Fig. 3. Various Filter Values for the Bounds Filter

filter is more important than the left side of the filter.

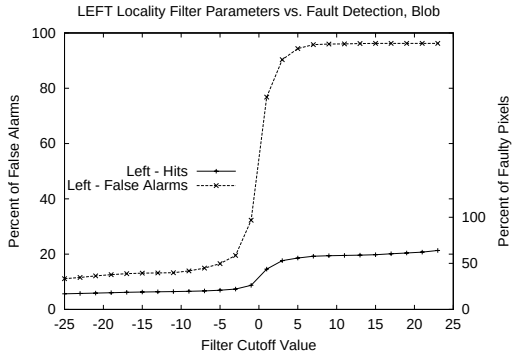
If we were expecting to see only these kinds of faults, then the filters could be further refined to reflect this. By looking at Tables I and II, which tabulate the total system-wide fault detections and false alarms, and then looking back at Tables III and IV, the rate of detection per side can be further quantified.

The balance between dependence on the left and right sides of the filter would be difficult to determine mathematically. Even if we could reduce the curves to their component formulae, the number of variable factors makes finding an optimal point a laborious task. At this point, it is easier to narrow the curve to an approximate area where we suspect the optimal balance would occur, and search for the desired tradeoff between false alarms and misses. If we want a minimum of some percentage of fault hits, we could look at a table such as Table I, and find the configuration of left and right filters at which this happens. We can then use a table (such as Table II) to find which of these points has the least number of associated false alarms. Once we determine the best point, Tables III and IV can be used to find which side of the filter is producing which percentage of the false alarms and hits.

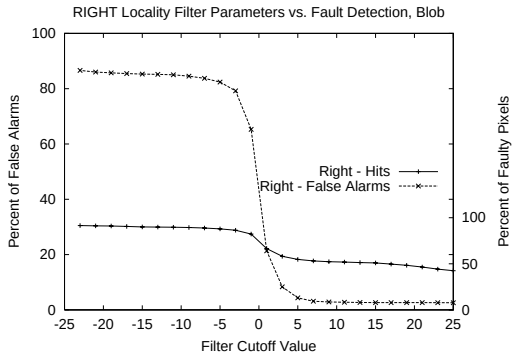
As an example, if an 80% fault detection rate is desired, we would first consult Table I. We would like the configuration of the left and right filters where an 80% detection rate is achieved. Because detection sensitivity and false alarm rate are directly proportional, finding the minimal config-

Cutoff	Left Only Hits	Right Only Hits
302	8.7%	91.2%
304	8.7%	91.2%
306	8.8%	91.1%
308	8.8%	91.1%
310	8.8%	91.1%
312	8.8%	91.1%
314	9.0%	90.9%
316	12.1%	87.8%
318	14.0%	85.9%
320	16.6%	83.3%
322	18.9%	81.0%
324	23.0%	76.9%
326	28.3%	71.6%
328	34.7%	65.2%
330	43.1%	56.8%
332	48.5%	51.4%
334	50.0%	49.9%

TABLE III
PERCENT OF FAULT HITS BY SIDE-PARTICULAR BOUNDS FILTERS



(a) Locality Filter (Left Cutoff)



(b) Locality Filter (Right Cutoff)

Fig. 4. Various Filter Values for the Locality Filter

Cutoff	Left Only False Alarms	Right Only False Alarms
302	0.0%	100.0%
304	0.0%	100.0%
306	0.0%	100.0%
308	0.0%	100.0%
310	0.0%	100.0%
312	0.3%	99.6%
314	3.9%	96.0%
316	11.0%	88.9%
318	18.6%	81.3%
320	24.1%	75.8%
322	31.7%	68.2%
324	41.0%	58.9%
326	51.3%	48.6%
328	67.3%	32.6%
330	87.9%	12.0%
332	98.5%	1.4%
334	100.0%	0.0%

TABLE IV
PERCENT OF FALSE ALARMS DETECTED BY SIDE-PARTICULAR BOUNDS FILTERS

uration which satisfies our fault detection criteria should also yield the least false alarms. Some example points of this in Table I happen with the left and right filters at (302K,325K), (304K,325K), (306K,325K), (316K,327K), and (318K,327K). Referencing these same points in Table II, we find that these points have associated false alarm rates of 53.7%, 53.7%, 53.7%, 51.7%, and 59.3% respectively. Among these, we find that the minimum false alarm rate happens with the left filter at 316K and the right filter at 327K. Referencing Table III we see that with this configuration the left filter is detecting 12.1% of the total faults, and right side is detecting approximately 73%. Referencing Table IV we also find that the left filter is responsible for 11% of the maximum possible false alarms, while the right side is yielding approximately 40%. If our goal was to minimize false alarms, we may have to be a little more lax on our expectation of fault detection with this filter.

We now have an approximate setting for which we can use this filter. Looking at the results, we can see that while the detection rates are acceptable, our false alarm rate is too high. If we had only this filter, it would have to be accepted as the price of such high fault detection. Luckily, the process is not yet complete - there is still the locality filter to apply. The locality filter itself will detect some of the same faults of the bounds filter, as well as having the

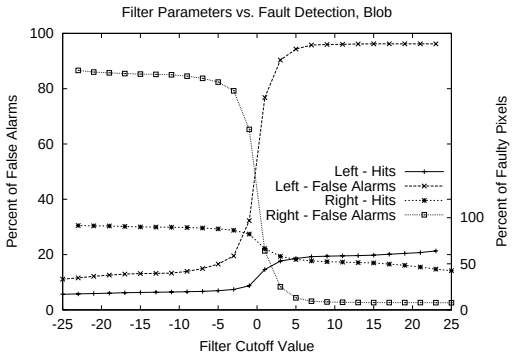


Fig. 5. Various Filter Values for the Locality Filter

Union of Filters, False Alarms Cols = Spatial Locality Filter Rows = Bounds Filter			
	60%	**70%**	**90%**
40%	15.7%	22.6%	76.0%
50%	15.7%	22.6%	76.0%
60%	15.7%	22.6%	76.0%
70%	36.3%	42.2%	84.2%
80%	59.9%	64.6%	90.5%
90%	77.1%	79.0%	94.5%

TABLE V
False Alarms with a Union of Filters

Union of Filters, Fault Hits Cols = Spatial Locality Filter Rows = Bounds Filter			
	60%	**70%**	**90%**
40%	63.7%	71.9%	89.6%
50%	64.0%	72.1%	89.7%
60%	67.5%	72.7%	90.2%
70%	76.3%	80.1%	94.2%
80%	84.1%	87.4%	96.8%
90%	93.0%	94.3%	98.7%

TABLE VI
Fault Hits with a Union of Filters

potential to detect faults which have been missed. With the union of the two filters, we can achieve good fault detection with few false alarms. The detection characteristics of each side of the locality filter can be seen in Figure 4. The union of the two sides of the locality filter is seen in Figure 5

The above results reflect only one set of output, the "Blob" dataset. As this is typical data in both bounds and locality, the filter settings should be equally efficient for any moderate case. Applying this exact configuration to the "Stripe" dataset affords an 80.6% detection rate for faults, but also incorrectly identifies 45.7% of possible false alarms. These results are approximately the same as those returned from the "Blob" dataset. If the filters are applied to the "Spots" dataset, it is found they will only result in a 69.5% fault detection. While this is a drop from the desired 80%, the false alarm overhead associated with this is only 18%. In order to achieve the fault detection desired, the filters for "Spots" need to be set at 318K and 323K. The reason for this can be seen when inspecting Figure 1(a); The most frequent data values for the "Spots" are slightly less than those in the other two datasets. This is a shortcoming of the simple absolute bounds filter - even a constant deviation in range will result in many incorrect identifications. This is also a good reason why a single application shouldn't rely solely on one filter for all of its fault detection. While the bounds filter does a good job for the average case, it may fail in extreme cases. By expecting less coverage from a single filter and relying on the cumulative effects of many filters, a wider spectrum of datasets can be tolerated.

The question remains, however, as to what degree of error detection can be expected from each filter. Just as the left and right bounds of each filter have unique characteristics, so do each of the two filters. A particular filter may increase in effectiveness consistently as its span is increased, or may only be increasingly effective for some initial range. Unfortunately, these filters do not combine in a cumulative fashion, as the left and right boundary filters do. Both filters may detect all of the same faults, or distinct faults, depending on the type and magnitude of the faults. The only way to determine the optimal balance between these filters is to repeat the method previously described for each filter, and find the best point of balance between those configurations.

For each filter, the "optimal" configurations for various levels of detection accuracy were configured. The spatial locality filter, with 60%, 70%, and 90% detection accuracy, was run alongside the bounds filter from 40% to 90% accuracy. The union of these two filters, with some configurations, allowed for detection greater than each of the individual filters - showing that some errors were unique to each filter's criteria. It was observed, though, that many of the faults were identified by both filters. Finding filters which are more strongly disjoint would be beneficial in this case. The detection attributes of the union of filters are similar for both other data sets as well, with the same relative results as the single bounds filter. Where the "Stripe" data set has approximately the same fault detection and false alarm characteristics of the "Blob" set, the "Spots" set tends to have lower fault detection success, but also significantly lower false alarm occurrences.

By using Tables V and VI, the tradeoffs between detection accuracy and false alarms can be analyzed for the "Blob" dataset. These tables show the percentage of incurred false alarms out of the maximum potential false alarms and fault detections out of the maximum faults which could be detected, respectively, for the union of the two filters. The number of faults which are disjoint between the two filters can also be estimated by seeing how many "extra" faults are detected by adding the second filter.

V. Results

By using the suggested method, the union of multiple filters, each with unique configurations, can be calibrated to achieve optimal results. Our final table reveals some characteristics of the two filters: namely that the number of disjoint fault detections between the two is very low. While this is not beneficial insofar as the creation of filters for OTIS, the fact that this information can be determined at all encourages the creation of new filters, for OTIS as well as for other applications.

In the case of OTIS, each filter is approximately of the same complexity. If it were the case that one filter was significantly more complex than the other, our decision could be weighted to reflect not only the results, but also the cost of filter runtime. The weighted table would allow the user to pick the best calibration of filters considering fault detection characteristics as well as calculation overhead.

VI. Conclusion

The focus of this paper has been the creation of a method by which "filters" (application specific data acceptance tests) can be calibrated. Faulty data may occur in a number of ways, one of the most elusive is as corrupted data. Such data may run through an application without any hint of its faulty nature - unless the application employs fault tolerance. Filters employ the observed data trends of nonfaulty data to distinguish which data is probably nonfaulty, and which may be faulty.

A filter must be calibrated such that the data that is judged as faulty or nonfaulty is actually so. Doing this requires the filters to know certain "cutoffs" beyond which the data is ambiguous, or definitely faulty. Adjustment of these cutoffs yields returns and penalties in the form of real faults detected and false alarms, respectively. It is at the user's discretion to balance these two.

A single application may have multiple filters, and it is necessary to be able to determine the tradeoffs of all filters working in unison. Fault detection and false alarm rates increase proportionally with respect to each other. If each filter can detect a small, disjoint set of faults, the overall false alarm rate may remain less than if a single filter were to detect the entire set of faults.

This method is critical for employing ALFTD. The application, using the filters, relies on a low-overhead, reasonably accurate method of fault detection to determine when redundant processes should be run. By employing this method to make these filters as accurate as possible, ALFTD itself can run with less overhead, and produce good fault tolerance.

References

[1] R. Ferraro, "Remote exploration and experimentation project plan," Tech. Rep., Jet Propulsion Lab, July 2000.

[2] E. Ciocca, "Application-level fault tolerance and detection," M.S. thesis, University of Massachusetts Amherst, 2002.

[3] K.-H. Huang and J.A. Abraham, "Algorithm-based fault tolerance for matrix operations," *Proc. IEEE Int'l Conf.*, pp. 518–528, jun 1984.

[4] J.C. Knight and N.G. Leveson, "An experimental evaluation of the assumption of independence in multiversion programming," *IEEE Transactions on Software Engineering*, vol. 12, no. 1, pp. 96–109, January 1986.

[5] J. Haines, V. Lakamraju, I. Koren, and C.M. Krishna, "Application-level fault tolerance as a complement to system-level fault tolerance," *The Journal of Supercomputing*, vol. 16, pp. 53–68, 2000.

[6] B. James, O. Norton, and M. Jr, "The natural space environment: Effects on spacecraft," 1994.