Computer Science • 18 (1) 2017 http://dx.doi.org/10.7494/csci.2017.18.1.3

Marcus Hilbrich
Markus Frank

ANALYSIS OF SERIES OF MEASUREMENTS FROM JOB-CENTRIC MONITORING BY STATISTICAL FUNCTIONS

Abstract *The rising number of executed programs (jobs) enabled by the growing amount of available resources from Clouds, Grids, and HPC (for example) has resulted in an enormous number of jobs. Nowadays, most of the executed jobs are mainly unobserved, so unusual behavior, non-optimal resource usage, and silent faults are not systematically searched and analyzed. Job-centric monitoring enables permanent job observation and, thus, enables the analysis of monitoring data. In this paper, we show how statistic functions can be used to analyze job-centric monitoring data and how the methods compare to more-complex analysis methods. Additionally, we present the usefulness of job-centric monitoring based on practical experiences.*

Keywords job-centric monitoring, monitoring, similarity, series of measurements, statistical functions, grid, cloud, analysis

Citation Computer Science 18 (1) 2017: 3–19

1. Introduction

The rising demand for computing time by scientists from different fields of research is an ongoing trend. This demand has been answered with more and more powerful computing systems. Nevertheless, the demand for further resources persisted, so the number of resources (like available CPU cores) has increased dramatically over the last several years.

To enable easy resource usage for such systems, a set of techniques was introduced. Such techniques are e.g., Portal-Systems, Grid and Cloud services. These techniques enable scientists with moderate knowledge about computer science to use huge amounts of resources by providing easier access. A drawback of the techniques is the decreased observability of the executed processes (which we call jobs). The reasons for this are the many introduced abstraction layers such as middlewares, batch systems, service layers, virtualization, etc. – where each layer hides information to allow for easier usage. As a result, it is unclear how efficient the resources are used, and silent errors during job execution remain mainly unobserved.

One solution dealing with the additional layers of job execution is job-centric monitoring; this offers online job observation and automatic post mortem analysis. In [11, 12], we showed how to build an infrastructure to handle job-centric monitoring data for huge installations. The analysis of monitoring data is a common Big Data challenge. Therefore, we started by studying related work [14], where we identified a set of analysis techniques like genetic algorithms [6, 7, 17, 29], machine learning [2, 19–21], sequence comparison [1, 9, 24, 25], intrusion detection [5, 26, 27], and statistic of events [4, 18, 28]. The most promising technique is a similarity comparison [8, 13]. However, this method is complex and computing-intensive.

Thus, we need to check how less-computing-intensive analysis performs and whether it is possible to execute a pre analysis to reduce the number of jobs that must be handled by more-complex algorithms. The analysis of series of measurements from job-centric monitoring works in principle like the following: for a group of jobs that are expected to have similar behavior, a reference is defined. Such a group of jobs can be based on a user running similar types of applications or of a specific application or service that is executed by different users. In each scenario, varying input data is used. Jobs behaving the same as the reference are marked as error-free execution. Outliers must be analyzed further.

In the following section – Section 2 – we give examples of two typical usage scenarios where job-centric monitoring can be helpful. A description of the domains and environments for the examples is also given. Afterwards. we present the fundamentals of job-centric monitoring – Section 3. In Section 4, we explain the test data based on two so-called basic jobs. To test the detection potential of the analysis methods, we varied the basic jobs by applying aberrations. The aberrations represent possible faults in job execution or changes in the execution environment. Afterwards, we describe the statistical functions and exemplify the potential of detecting faults in job execution in Section 5. The description is based on basic jobs and the applied

aberrations. An outlook to further test data is given in Section 5.6. To classify the detection potential of the statistical functions, we present a comparison of an analysis method based on similarity functions in Section 6. Section 7 refers to the examples from Section 2 and demonstrates the usefulness of job-centric monitoring in three use cases. In the last section – Section 8 – we give a conclusion and point out further work.

2. Domains for job-centric monitoring

In the following, we describe domains that can benefit from job-centric monitoring. In section 7, we give more-detailed information for real-world examples of this domain. The first domain is the German Grid infrastructure (one of our customers we supported as resource provider). As a resource provider, we allowed other users to access computing and storage systems. Therefore, we provided access via different grid middlewares; e.g., Globus[1]. The grid middlewares used a mapping mechanism to map grid users to generic/internal users of the computing center. This is needed to submit a user's job to the batch system. The batch system cares about allocating hardware for the user and moves the user's job to the operating system of the allocated hardware.

Furthermore, this is needed to invoke an additional abstraction layer; therefore, we see a lot of abstraction layers just to start a job. It is clear that the job needs computing resources (hardware) and an operating system. In addition, most jobs need some software libraries and interpreters like Perl or Java. In our example, the job wants to run a special program to simulate parts of the cardiovascular system to prepare for a medical operation. This program is maintained by a group that builds services for health personnel. The developers of the program make up a scientific research group. The services for health personnel is based on additional hardware (not maintained by us as grid resource provider) and additional layers of software.

Back to our example; to enable very easy access for the health personnel, web browser-based access to a portal is provided. The portal allows the user to access different storage locations on the grid for uploading data and accessing different programs (like the one from above) as well as a workflow editor for combining multiple programs (e.g., transforming the uploaded medical data, so that it can be used as input for a fluid dynamic's simulator, to run multiple simulations for the cardiovascular system of a patient, and interpreting the results).

To sum up our small example, we have a lot of different abstraction layers and different groups of people involved in the process; thus, the system is quite complex and error-prone. A missing library, a poorly configured workflow, or invalid input data can stop the system from working properly. And a completed workflow execution dose not mean the absence of silent faults or near-optimal usage of the resources. A way to make such a complex system more transparent is job-centric monitoring.

[1] `http://toolkit.globus.org/toolkit/`

The second example comes from particle physics. The principle is similar to the one before, but only a single program was used (no complex workflow and fewer abstraction layers were introduced). The used libraries and program were installed and maintained by us, and we had direct contact to the users; thus, we had more control over the complete execution of the program and direct contact to the physicists that executed the program via the middleware installed on our computing systems.

The testing data (provided later) is also based on this program, so we will give more details in Section 4 by explaining the basic jobs.

3. Fundamentals of job-centric monitoring

Job-centric monitoring was introduced as a grid-based monitoring system called AMon [22]. AMon uses the monitoring infrastructure SLAte [11, 12] to store data in an scalable manner.

The monitoring data, can be recorded in variable time intervals, ranging from seconds to minutes. This was based on the experience that many jobs failed during their starting phases. Thus, the intervals were set shorter for each starting phase. Over the last few years, we discovered that the number of early failing jobs dramatically decreased, so we decided on a constant measurement interval (which is easier to handle for automatic analysis processes).

The monitoring data is recorded directly on the computing node. Used are common monitoring techniques [23] known from tools like top or ps without privileged access rights. The recorded data contains information about a job; e.g., consumed CPU time, CPU load, main memory, and access to the file system. Also recorded is information about the executing system like the free main memory, state of the storage systems, used network bandwidth, and number of interrupts. In addition, scheduling information such as time, date, and wall-clock time of the job are recorded. Each of the measurements is directly transferred to the SLAte infrastructure and can be accessed by users with the needed privileges (e.g., the user who started the job).

AMon is used [13] for visualizing and analyzing the monitoring data. AMon can be used as a desktop application or can be provided by a server and accessed via a web browser. For a single user, it is often needed to present thousands of jobs. As an answer to the demand, the monitoring data was presented as color coded graphs (for example). The visualization allowed us to compare at least dozens of jobs by manual analysis. It was also noticed that the manual analysis needed a lot of experience from the users as well as a noticeable amount of time. To reduce the time for manual analysis, an automatic analysis was developed. Parts of the complex analysis process are presented in this paper. By the way, the automatic analysis also reduces the need of visualizing thousands of jobs (which is currently not possible with AMon and an unsolved challenge in the related disciplines). Based on the automatic analysis, the user is only confronted with jobs that do not behave as expected and likely had problems during execution. To identify jobs that behave unexpectedly, a reference job is needed. A reference can be

based on a controlled and error-free run of the application or service. Another – not yet realized variant – is to base the reference not on historical information but on a behavior description from a model-based software development process. After the first reference is defined, jobs can be analyzed by comparing them to the reference. In case a job and a reference have similar behavior, the job is marked as normal execution and no further – manual – analysis is performed. In case a job does not match the behavior of the reference, a further analysis is necessary. After this manual analysis, the initial reference (for well-behaving jobs) can be updated or a reference for the specific error of execution (e.g., over-utilization of the hardware) can be set. In this case, the error can be automatically detected in the future.

We also build a test system to inform users directly by mail about the executed jobs and potential failures upon execution.

4. Test data

From a larger dataset, we selected test cases that are suitable to demonstrate the principle approach of the analysis by statistical functions. Information about additional test data is given in Section 5.6. In the following, we will use two synthetic basic jobs. The first one is based on the CKM-Fitter [3, 15] application. The series of measurements is plotted in Figure 1. In concrete, the CPU load over a runtime of about eight hours is shown. At the beginning of the job, CPU usage increases from 0 to 1. This is the starting phase of the job. Afterwards, a working phase follows with a constant value of 1, which means that one CPU is used at 100% load. In the last phase, the job ends with decreasing CPU-usage. There is no measurement with zero percent CPU usage at the end of the job. This is based on the fact that the job became deallocated before such a measurement was taken. An example of such a behavior is an application that reads in data at the beginning, does some extensive calculations, and outputs some data at the end. Based on the fact that a job is not limited to a single sequential application, multiple CPUs can be used. This can lead to a CPU load of more then 100%.

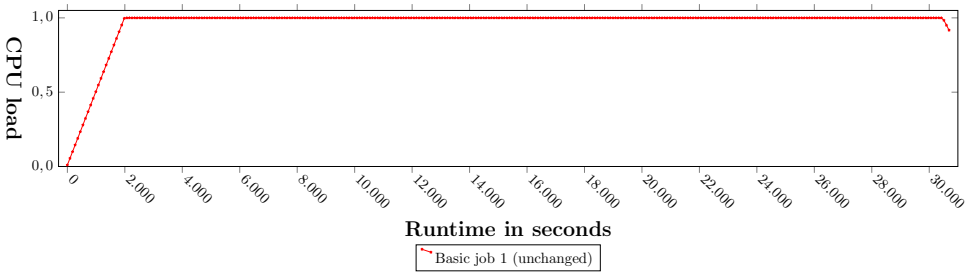

Figure 1. Plot of the used monitoring data (CPU load) of basic job 1.

To exemplify aberrations, we constructed additional jobs based on basic job 1. In the following, so-called gaps are applied. A gap changes the monitored values within a defined time interval. In our example, the value is varied by a load of 1 over 10% of the runtime, either as an increase *(<+<)* (Fig. 2) or a decrease *(<–<)* (Fig. 3). In addition, we use a gap that increases and subsequently decreases the values. The gap is called *(<+–<)* and is shown in Figure 4.

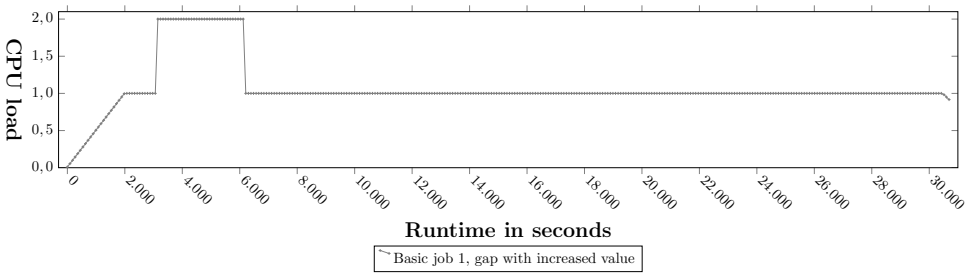

Figure 2. Adaptations of basic job 1 with gap *(<+<)*.

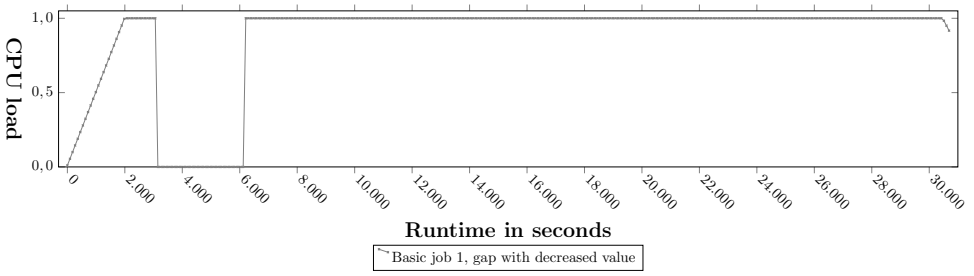

Figure 3. Adaptations of basic job 1 with gap *(<–<)*.

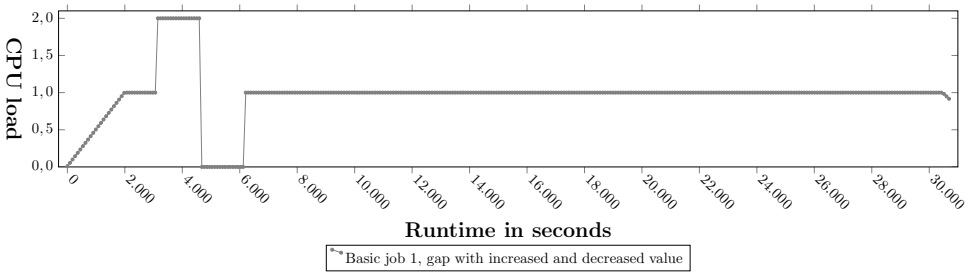

Figure 4. Adaptations of basic job 1 with gap *(<+–<)*.

For demonstration purposes, we introduced a second synthetic job (basic-job 2), shown in Figure 5. The job consists of 16 CPU-intensive working phases, each followed by a waiting phase for network or other I/O operations without CPU demand. The working phases take $\frac{2}{3}$ of the job, and the waiting phases are $\frac{1}{3}$. An adaption of basic job 2 is shown in Figure 6. This job was executed by a system with double CPU speed and the same I/O bandwidth. Thus, the working phases need only half the time as compared to basic-job 2, while the waiting phases stay the same. A change of CPU changes the execution of a job, but it is not a considerable aberration. Thus, a detection algorithm should not present this adaptation as an error. Similar adaptations are caused by various influences, like an additional iteration of an executed loop or system noise [16].

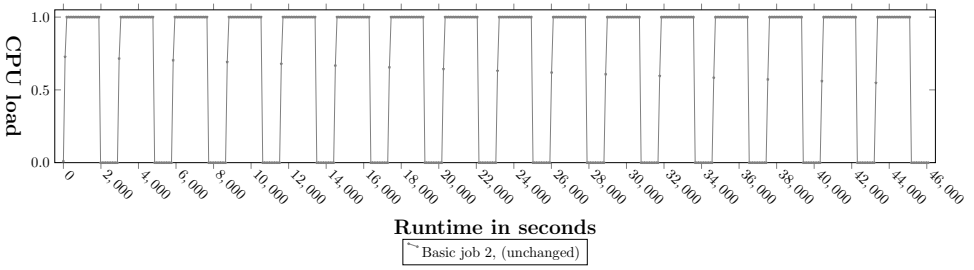

Figure 5. Plot of the used monitoring data (CPU load) of basic job 2.

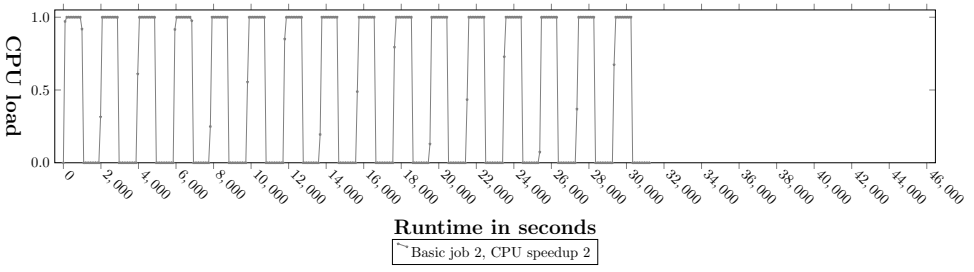

Figure 6. Adaptations of basic job 2 with speedup of 2 on CPU-intensive parts.

5. Statistical functions

One of the advantages of statistical functions is the short computing time as compared to other analysis methods. A previously described analysis method [13] based on similarity functions requires an alignment with complexity up to $O(n^5)$. For calculating statistical functions, we need to read the series of measurements once and perform some calculations per measurement point (e.g., cooperation to find the minimum or

summing up to calculate the mean). Thus, the computational complexity is in the class of $O(n)$.

In concrete, we analyzed the median, minimum, maximum, mean, and standard deviation of a measurement variable (e.g., CPU load) for a single job. Each statistical value will be explained and demonstrated in the following sections. An overview about the results is given in Table 1.

Table 1
Comparison of analysis methods.

	Median	Mini-mum	Maxi-mum	Mean	Standard deviation	Similarity functions
Handles variable measurement intervals:	no	yes	yes	yes	yes	yes
Gap, increasing maximum value:		no	yes	yes	yes	yes
Gap, increasing value during waiting time:		no	no	yes	yes	yes
Gap, decreasing values:		no	no	yes	yes	yes
Increases and decreases of values do not compensate:	yes	yes	no	no	yes	yes
No false positives by timing changes:	yes	yes	yes	no	no	yes

5.1. Median

The median is the measurement value for which half of the additional measurement values are the same or higher and the other half are the same or smaller. In case the number of measurement values is even, the median is based on the mean of two neighbored values.

For job-centric monitoring, the time distance between measurement values is not constant. So, the result of determining the mean depends on the timing when measurements are taken. As result, the same jobs with different measurement timestamps result in different determined medians. Thus, the median can change even when the job and reference have the same behavior. One reason to use miscellaneous intervals is to achieve higher accuracy for some parts of a job; e.g., for the error-prone starting phase. Based on these considerations and some preliminary tests, we removed the median from future investigations.

5.2. Minimum

The minimum of a series of measurement is often zero (like for the basic job in Figure 1). This is plausible because a first measurement is often done before the monitored job starts to consume CPU resources or memory. Negative consumption of such working resources is impossible; thus, it is impossible to go below the minimum of

zero. Under such conditions, the detection potential for the minimum is nonexistent. Independent of an applied gap, the minimum stays at zero.

5.3. Maximum

The type of analyzed and executed program often defines the highest value of the series of measurements. CKM for instance uses one CPU at full load, so the maximum is a CPU load of one for a non-faulty execution (reference). Whenever the use of a working resource exceeds this limit, it can be easily detected. Thus, a gap increasing the value (like *(<+<)* in Figure 2 and *(<+–<)* in Figure 4) can be easily detected. In both examples, the maximum is increased from 1.0 to 2.0. Aberrations that only cause decreases (like *(<–<)* in Figure 2) cannot be detected (the maximum stays at 1.0). It is also clear that increases can only be detected when they increase the maximum value; so, if we have (for example) increased usage of the CPU in a waiting phase (basic job 2, Figure 5) but the load stays below the maximum, we cannot detect it.

The maximum is just the value of the highest measurement, so it is not dependent on the concrete time of the measurements. Thus, a partial speedup (like that shown in Figure 6) does not change the maximum.

5.4. Mean

In case the measurements are equidistant, the mean x_{eqe} could be calculated by the following formula (where x_i is the measurement value with index i and n the number of measurements):

$$\bar{x}_{eqe} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{1}$$

The previous formula is only valid for equidistant measurement intervals. For variating measurement intervals, the formula has to be extended for calculating the mean for series of measurements of job-centric monitoring data x_{job} based on timestamp t_i at which a measurement was taken. In concrete, a value is to be considered for half the measurement interval before and after the measurement:

$$\bar{x}_{job} = \frac{1}{t_n} \cdot \left(t_1 \cdot x_1 + (t_n - t_{n-1}) \cdot x_n + \sum_{i=2}^{n-1} (t_{i+1} - t_i) \cdot x_i \right) \tag{2}$$

The mean for basic job 1 (Fig. 1) is 0.97. In case the values are either increased or decreased, the aberration can be detected. The increase by *(<+<)* in Figure 2 changes the mean to 1.07, and the decrease by *(<–<)* gives a mean of 0.87.

A drawback is that increases and decreases can compensate each other. This is demonstrated by gap *(<+–<)* (in Figure 4). The mean is 0.97 (the same as the reference), so the aberration to basic job 1 cannot be detected.

In comparison to the maximum, other aberrations can be detected. It is not needed to extend a certain limit, and it is possible to find increases and decreases. Nevertheless, a combination of increases and decrease leads to the compensation of

both types of apparitions. In the worst-case scenario (like the example above), an error can not be detected at all by calculating the mean.

The calculation of the mean also depends on the times of the measurements. To demonstrate the effect, we use basic-job 2 (Fig. 5) and the partly sped-up job from Figure 6. The mean changes from 0.65 to 0.48. Thus, the change in the job cannot be distinguished from a fault. In case of a negative speedup (for example), the mean increases (speedup of 0.5 results in a mean of 0.78). So, a faulty change (demonstrated by the gaps) and non-faulty change (demonstrated by partial speedup) can both decrease and increase the mean. Thus, both effects can also compensate each other. Based on these findings, we expect the mean as an unpredictable value to identify faulty jobs.

5.5. Standard deviation

The standard deviation cannot be calculated based on samples – by definition. However, for comparing different jobs to a reference, we can determine an empirical standard deviation that is based on a set of samples that can be given by a series of measurements; e.g., job-centric monitoring data of a single job. The definition of standard derivation σ_X is:

$$\sigma_X = \sqrt{Var(X)} \tag{3}$$

Where $Var(X)$ is the variance of the series of measurement, which is defined as:

$$Var(X) = E\left(\left(X - E\left(X\right)\right)^2\right) \tag{4}$$

Where E(X) is the expectation – for the empirical standard derivation, the expectation is identical to the mean that was used in the last section.

Based on the formulas, the empirical standard derivation can be calculated for the test data. For basic job 1 (Fig. 1), we get a value of 0.14. After applying the gap *(<+<)* (Fig. 2), the derivation changes to 0.36. By applying gap *(<–<)*, the derivation changes to 0.34. Even the gap for a combination of an increase and a decrease *(<+–<)* (Fig. 4) results in a derivation that can clearly distinguish from the basic job. In concrete, the derivation is 0.36.

The influence of the timing of a job can also be demonstrated. The reference – basic-job 2 (Fig. 5) – gives a derivation of 0.47. In case the CPU-intensive parts of the job are sped up (Fig. 6), the derivation is 0.49. Thus, a change in the timing of a job that is not based on a fault changes the derivation, so non-faulty changes can lead to false positives.

5.6. More-detailed tests

The exemplification of the potential of statistical functions for analyzing job-centric monitoring data has already been shown, and a summary is offered in Table 1. Nevertheless, for the underlying investigations, a much-wider test set was used. A part of the investigations was set on basic job 1. Besides the three different introduced gaps,

an additional gap that first lowers and then increases the value was used. The gaps were verified in the intensity of time and value. Also, the influence of the number of applied gaps as well as the position was investigated.

The influence of the counted gaps and their positions can be analyzed by plots like Figure 7. The figure shows that the number of applied gaps and position have no relevant influence on the calculated standard derivation. In this case, basic job 1 was used as reference.

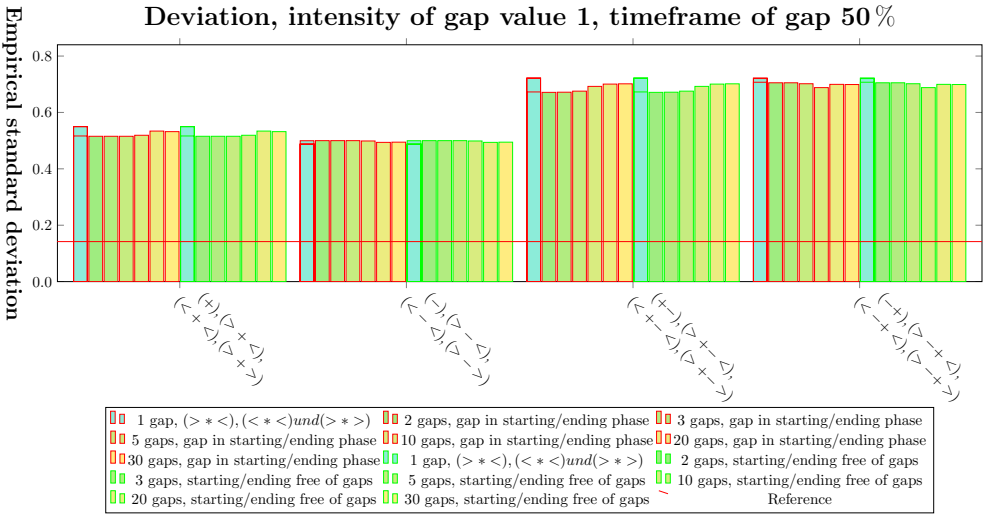

Figure 7. Empirical standard deviation for jobs based on basic job 1 with a very strong gap applied

An even-more-detailed plot that only increases the value is given in Figure 8. The five groups show that an increased time span of the gap result in an higher aberration of the standard deviation value. Within each of the five groups, it can be shown that a higher change in the measurement value leads to a higher aberration of the calculated standard deviation. The same results are also valid for the other gap types.

Additional diagrams for the other statistical functions can be plotted and analyzed. A complete set of these plots is printed in dissertation [10]. Based on this testing data, we can summarize our findings in Table 2.

In addition to basic job 1, we also investigated on a basic job including a waiting phase where no CPU load is caused. This tests verified the findings from Section 5.3. Also, different time spans for taking measurements are analyzed. These tests support our statements from Section 5.1. An additional wide test set was built on the second basic job. This job was used to analyze different kinds of time drifts. The changed CPU speed like that shown in Figure 6 is only an example for such tests. Based on the fact that the additional tests did not point out additional principle findings

and the illustration of the data took dozens of pages in [10], we decided to only refer to [10] here.

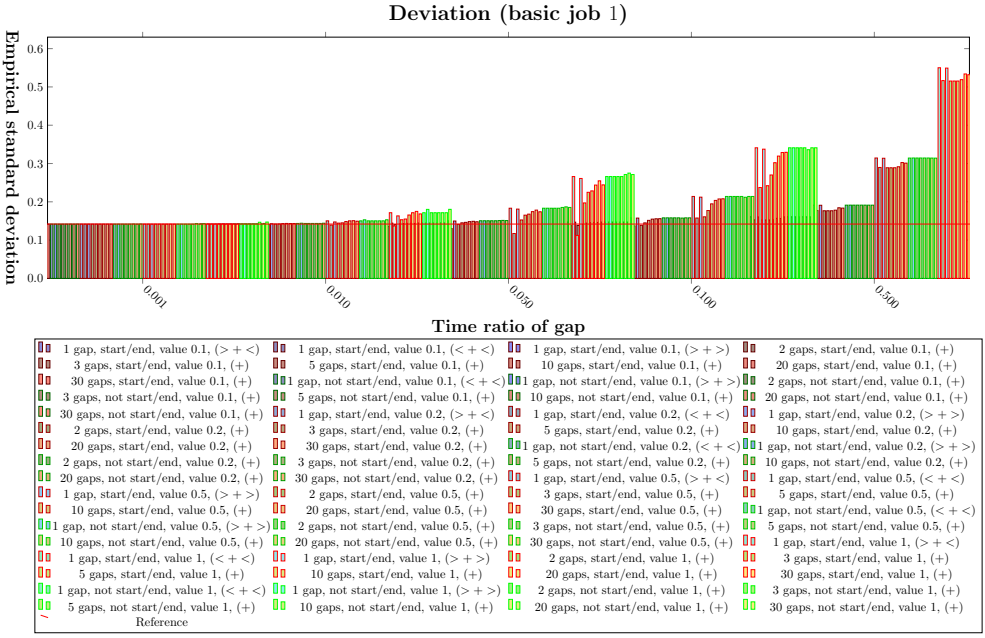

Figure 8. Empirical standard derivation for jobs based on basic job 1 with different gaps increasing measurement values (CPU load)

Table 2

Additional findings for the analysis methods based on basic jobs 1 and 2.

	Maximum	Mean	Standard deviation	Similarity functions
Mainly independent of position of gap:	only basic job 1	yes	yes	yes
Mainly independent of number of gaps:	yes	yes	yes	yes
Longer gaps are indicated by higher aberration to reference:	no	yes	yes	yes
Higher value change of gaps is indicated by higher aberration to reference:	no	yes	yes	yes

6. Comparison to similarity functions

In this paper, we focus on statistical functions. Nevertheless, we have to include more-complex methods for analyzing job-centric monitoring data to get a better understanding of the limitations and advantages of statistical functions. The most-

promising methods we found and developed further are based on similarity comparison. The first steps in using the cross correlation function to compare job-centric monitoring is shown by [13]. Further developments have already been done; so besides the local and global normalization and linear time adaption explained by [13], an automatic adoption to time drifts based on an optimization strategy using educated guessing was developed. The optimization strategy is currently prepared for publication for a wider audience. An initial description can already be found in dissertation [10]. Based on this dissertation, we added the possibilities of detecting aberrations of job execution to Table 1 and Table 2.

By considering Table 1 and Table 2, it is clear that the standard derivation gives the best results as compared to the analyzed statistical functions. Based on the standard deviation, the different applied aberrations presented here (and even other papers) can be detected. The same aberrations can be detected by similarity comparison. The difference between the two methods is that the standard deviation cannot distinguish between an error during job execution and an acceptable change of timing by changed input data or different execution environments, for example.

7. Benefits of job-centric monitoring

It is hard to present a valid measurement of the fraction of found faults. This is based on the fact that not all faults are known, and even manual analysis is no guarantee for identifying each error. To show the helpfulness of job-centric monitoring and automatic analysis, we reactivate the examples from Section 2.

In one of our examples, we collected monitoring data for the particular physics at different times. During one of the periods, we discovered instabilities. The automatic analyses were not yet established, so we discovered the problem by the symptom of a rising number of aborted jobs. In a step-by-step solution, we identified the effected nodes of our hardware and noticed that some of the system processes were aborted due to main memory limitations, so we rebooted the appropriate nodes. A later analysis based on job-centric monitoring data identified additional problematic nodes. The nodes executed jobs with a CPU load near zero, which was easily identified by the visualization after we knew what to look for. The underlying problem was the same as for the previously repaired node, but other system processes were aborted.

An automatic analysis of the monitoring data would helped a lot in this case. The effected jobs showed a dramatic change of the mean for CPU usage. Thus, the automatic analysis had presented the first irregular job, so we were aware of the problem much earlier and could fix it before losing a lot of CPU-hours by defective nodes.

In another example, we described the more-complex health care system. Users complained about long waiting times. The workflow operators identified a program in the workflow, that was sometimes aborted and re-executed. The program maintainers identified that the problem only occurs on some computing resources. So, as operators of one of the effected resources, we got involved. After identifying some of the effected jobs, we hunted down the symptom from layer to layer, which included reading the log-

files, and a lot of personal communication, we confirmed what happened. The failing jobs extended the limit of the reserved main memory. Thus, the batch system aborted the jobs; this was reported to the middleware, so workflow management registered the problem and re-executed the program. Each re-executions extended the waiting time dramatically. Even worse, a re-execution can once again be executed on a computing resources with to low main memory capacity, resulting in an additional re-execution.

With job-centric monitoring, the problem could easily be solved. The over-utilization could be seen directly by the enduser or the workflow developer. Thus, changes in the input data could be directly mapped to a higher demand of main memory, and the needed resources for a job could be adapted (this took quite a while to be realized, based on the long debugging process). Also, the number of involved parties would be reduced because it would not be necessary to contact the resource provider to analyze the job's behavior. An automatic analysis based, for example, on the maximum had been able to identify and point out the first accordance of increased demand on the main memory, so the problem could be fixed before the changed input data led to a problem and caused a long debug session.

8. Conclusion and future work

In our opinion, the ability of the analysis method using the similarity function makes a big change in the quality of analysis results. The potential of finding unusual behavior is high, and different kinds of silent faults can be detected. Thus, a more-efficient resource usage is enabled based on removing problems from the job-execution process. Nevertheless, checking the behavior of executed jobs is an additional task for most users. Thus the accepted effort and time spent is very low. This also demands, that a time spending manual analysis of a job is only acceptable in case it is really needed. To manually analyze just one job without an fault can be so frustrating, that job-centric monitoring will be used never again. Following, our key demand is to get a low rate of false positives to reach a high acceptance rate from the users.

On the one hand, it is clear that we prefer the more-complex analysis method based on similarity functions. The reason is the ability to avoid more false positives. On the other hand, the calculation of the standard deviation is much faster because the analysis is not so complex and, thus, less computing-intense. So, we want to test whether we can use the standard deviation as a first test and afterwards check the jobs with aberrations once again by similarity functions. Only aberrations confirmed by the second check are presented to the user. Based on the provided real-world examples, we have also seen that merely analyzing the maximal CPU and main memory demand could point out some irregularities and allow us to fix the underlying problem very easily.

An additional advantage of the methods based on the similarity functions is that it points out exactly where an aberration of a job to the reference is located. By using the standard deviation for only preselecting jobs, this advantage is conserved; however, computing demand for the analysis can still be reduced. Thus, we can get

the advantages of both methods in case the number of false positives given by the standard deviation stays moderate. This rate depends on the concrete usage scenario and has to be further analyzed in real-world examples.

Until now, we have focused on the theoretical impact of different analysis methods. As a next step, we want to establish cooperation with operating centers for scientific computing to further test and properly tune and adapt our analysis methods under real-world conditions. This allows a much bigger testbed that is not limited to selected applications or synthetic data. Furthermore, we could see and quantify the influences of users and applications as they change over time.

References

[1] Bellman R.: *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 2010.

[2] Chan P., Stolfo S.J.: Toward Parallel and Distributed Learning by Meta-Learning. In: *AAAI Workshop in Knowledge Discovery in Databases*, pp. 227–240, 1993.

[3] Charles J., Höcker A., Lacker H., Laplace S., Diberder F., Malclés J., Ocariz J., Pivk M., Roos L.: CP violation and the CKM matrix: assessing the impact of the asymmetric B factories. *The European Physical Journal C – Particles and Fields*, vol. 41(1), pp. 1–131, 2005, `http://dx.doi.org/10.1140/epjc/s2005-02169-1`.

[4] Denning D.E.: An intrusion-detection model. *IEEE Transactions on Software Engineering*, vol. 13(2), pp. 222–232, 1987.

[5] Dickerson J.E., Dickerson J.A.: Fuzzy network profiling for intrusion detection. In: *Proceedings of NAFIPS 19th International Conference of the North American Fuzzy Information Processing Society, Atlanta*, pp. 301–306, 2000.

[6] Dobai R., Balaz M.: Genetic method for compressed skewed-load delay test generation. In: *Design and Diagnostics of Electronic Circuits Systems (DDECS), 2012 IEEE 15th International Symposium on*, pp. 242–247, 2012.

[7] Grefenstette J.: Optimization of Control Parameters for Genetic Algorithms. *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 16(1), pp. 122–128, 1986.

[8] von Grünigen D.: *Digitale Signalverarbeitung: Mit einer Einführung in die kontinuierlichen Signale und Systeme*. Fachbuchverlag, Leipzig, 2008.

[9] Gusfield D.: Algorithms on Stings, Trees, and Sequences. *Computer Science and Computational Biology*, 1997.

[10] Hilbrich M.: *Jobzentrisches Monitoring in Verteilten Heterogenen Umgebungen mit Hilfe Innovativer Skalierbarer Methoden*. Dissertation, Fakultät Informatik der Technischen Universität Dresden, Germany, 2014.

[11] Hilbrich M., Müller-Pfefferkorn R.: A Scalable Infrastructure for Job-Centric Monitoring Data from Distributed Systems. In: M. Bubak, M. Turala, K. Wiatr, eds., *Proceedings Cracow Grid Workshop '09*, pp. 120–125, 2010.

[12] Hilbrich M., Müller-Pfefferkorn R.: Achieving scalability for job centric monitoring in a distributed infrastructure. In: G. Mühl, J. Richling, A. Herkersdorf, eds., *ARCS Workshops*, *LNI*, vol. 200, pp. 481–492, GI, 2012.

[13] Hilbrich M., Müller-Pfefferkorn R.: Cross-Correlation as Tool to Determine the Similarity of Series of Measurements for Big-Data Analysis Tasks. In: *2015 International Conference on Cloud Computing and Big Data (CloudCom-Asia)*, 2015.

[14] Hilbrich M., Weber M., Tschüter R.: Automatic Analysis of Large Data Sets: A Walk-Through on Methods from Different Perspectives. In: *Cloud Computing and Big Data (CloudCom-Asia)*, pp. 373–380, 2013.

[15] Höcker A., Lacker H., Laplace S., Le Diberder F.: A new approach to a global fit of the CKM matrix. *The European Physical Journal C – Particles and Fields*, vol. 21(2), pp. 225–259, 2001, `http://dx.doi.org/10.1007/s100520100729`.

[16] Hoefler T., Schneider T., Lumsdaine A.: Characterizing the Influence of System Noise on Large-Scale Applications by Simulation. In: *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pp. 1–11, IEEE Computer Society, Washington, DC, USA, 2010, `http://dx.doi.org/10.1109/SC.2010.12`.

[17] Holland J.: Genetic Algorithms. *Scientific American*, vol. 267(1), pp. 66–72, 1992.

[18] Lazarevic A., Ertoz L., Kumar V., Ozgur A., Srivastava J.: A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection. In: D. Barbará, C. Kamath, eds., *Proceedings of SIAM Conference on Data Mining*, 2003.

[19] Lee W., Stolfo S., Mok K.: A data mining framework for building intrusion detection models. In: *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, pp. 120–132, 1999.

[20] Lee W., Stolfo S.J.: Data Mining Approaches for Intrusion Detection. In: *Proceedings of the 7th conference on USENIX Security Symposium – Volume 7*, SSYM'98, pp. 6–6, USENIX Association, Berkeley, CA, USA, 1998, `http://dl.acm.org/citation.cfm?id=1267549.1267555`.

[21] Lee W., Stolfo S.J.: A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information and System Security*, vol. 3(4), pp. 227–261, 2000, `http://doi.acm.org/10.1145/382912.382914`.

[22] Lorenz D., Borovac S., Buchholz P., Eichenhardt H., Harenberg T., Mättig P., Mechtel M., Müller-Pfefferkorn R., Neumann R., Reeves K., Uebing C., Walkowiak W., William T., Wismüller R.: Job monitoring and steering in D-Grid's High Energy Physics Community Grid. *Future Generation Computer Systems*, vol. 25, pp. 308–314, 2009, `http://dx.doi.org/10.1016/j.future.2008.05.009`.

[23] Müller-Pfefferkorn R., Neumann R., William T.: AMon – a User-Friendly Job Monitoring for the Grid. In: T. Priol, M. Vanneschi, eds., *CoreGRID*, pp. 185–192, Springer, 2007.

[24] Myers E.W.: An O(ND) difference algorithm and its variations. *Algorithmica*, vol. 1, pp. 251–266, 1986.

[25] Needleman S.B., Wunsch C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, vol. 48(3), pp. 443–453, 1970.

[26] Paxson V.: Bro: a system for detecting network intruders in real-time. *Computer Networks*, vol. 31(23–24), pp. 2435–2463, 1999, `http://www.sciencedirect.com/science/article/pii/S1389128699001127`.

[27] Roesch M., Telecommunications S.: Snort – Lightweight Intrusion Detection for Networks. pp. 229–238, 1999.

[28] Smaha S.E.: Haystack: An intrusion detection system. In: *Proceedings of the IEEE 4th Aerospace Computer Security Applications Conference*, 1988.

[29] Tang K., Man K., Kwong S., He Q.: Genetic algorithms and their applications. *Signal Processing Magazine, IEEE*, vol. 13(6), pp. 22–37, 1996.

Affiliations

Marcus Hilbrich
s-lab – Software Quality Lab, Universität Paderborn, Zukunftsmeile 1, 33102 Paderborn, Germany, `marcus.hilbrich@uni-paderborn.de`

Markus Frank
Technische Universität Chemnitz, Straße der Nationen 62, 09107 Chemnitz, Germany, `markus.frank@informatik.tu-chemnitz.de`

Received: 16.11.2015
Revised: 9.06.2016
Accepted: 10.06.2016