Towards a Holistic Approach to Fault Management: Wheels Within a Wheel

Moises Goldszmidt (Mirosoft Corporation), **Miroslaw Malek** (Humboldt-Universität zu Berlin), **Simin Nadjm-Tehrani** (Linköping University), **Priya Narasimhan** (Carnegie Mellon University), **Felix Salfner** (Humboldt-Universität zu Berlin), **Paul A. S. Ward** (University of Waterloo), **John Wilkes** (Google).

Abstract

Systems with high dependability requirements are increasingly relying on complex on-line fault management systems. Such fault management systems involve a combination of multiple steps – monitoring, data analysis, planning and execution – that are typically independently developed and optimized. We argue that it is inefficient and ineffective to improve any particular fault management step without taking into account its interactions and dependencies with the rest of the steps. Through six real-life examples, we demonstrate this inefficiency and how it results in systems that either under-perform or are over-budget. We propose a holistic approach to fault management that is aware of all relevant aspects, and explicitly considers the couplings between the different fault management steps. We believe it will produce systems that will better meet cost, performance and dependability objectives.

1 Introduction

Large, complex systems frequently experience faults, and those faults need to be handled to limit the damage they cause. Fault management is the set of processes used to ensure dependability of the service, i.e., uninterrupted, reliable, secure and correct operation, at reasonable cost. An important subgoal in modern fault management approaches is to automate as much as possible to reduce human intervention and administration, which is expensive, error-prone and may even be infeasible in large systems. In turn, that requires a clear statement of the objectives and processes used to achieve the desired outcomes.

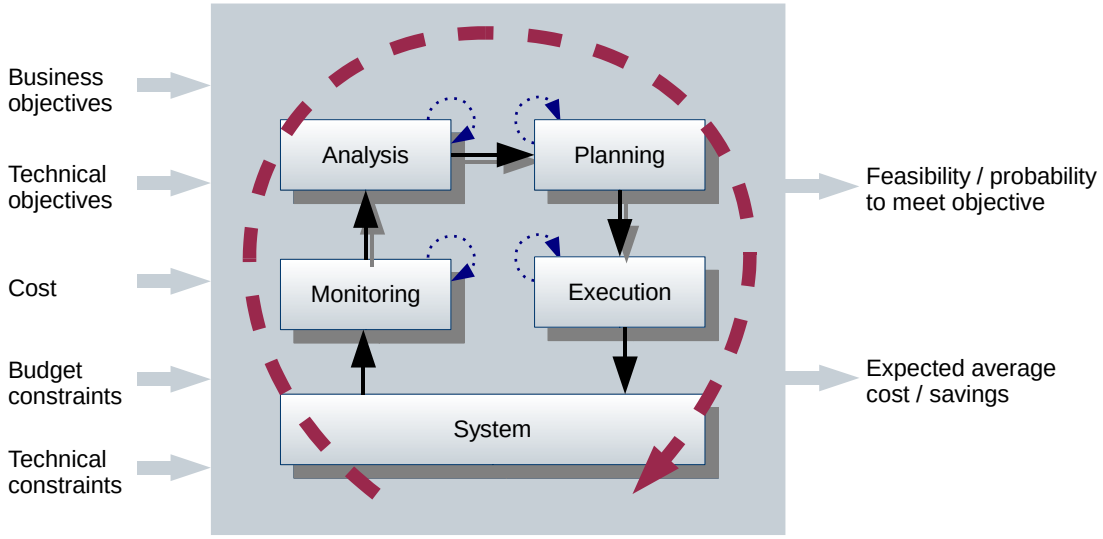

Figure 1. The loop of actions performed in fault management systems.

Fault management systems typically include many steps, which are often carried out sequentially, as shown in Figure 1: system monitoring, analysis of monitored data, planning of recovery strategies, and execution of mitigation actions. Although this taxonomy is a convenient compartmentalization, it is usually ineffective and inefficient to optimize a particular fault management step in isolation, without taking into account its interaction with the rest of the steps and how these interactions affect the overall objectives. That is, rather than asking "how should we improve a particular step?" a better question is "how should we configure the steps to maximize the overall benefit?"

For example, if the planning step offers only three possible recovery actions – say, to reboot a machine, reimage a machine, or call the operator – it is unnecessary for the analysis step to do anything other than to map the outcome of the monitoring step to one of these three actions. Any further effort in the analysis step is irrelevant as it has no bearing on the overall fault management outcome; indeed, such further effort might complicate the overall system (e.g., introduce bugs), waste cycles at runtime (which may impact availability), and waste developers' time on meaningless tasks.

We propose a holistic approach to the problem of deciding how much effort to invest where by addressing all four steps (monitoring, analysis, planning and execution) and their influences on each other, keeping in mind the main objectives – cost minimization and high availability. Our approach allows local optimization within the operational envelope of each fault management step, but links the global adaptation of these local optimizations to the (global) business goals of the system, resulting in a highly effective and coordinated combination of the four fault management steps. Therefore, we avoid local optimizations that do not help achieve the overall goal.

In support of our arguments, we present six real-life examples that explore the pitfalls of merely-local optimizations. The price of not having a proper focus on key objectives and ignoring the holistic approach is high and can no longer be neglected. This paper is a call to arms, to energize the community and give the impetus needed to overcome pitfalls of local optimization, and thereby improve the overall effectiveness and cost of systems.

Historically, academic research on automated fault management has been mainly driven by technical challenges, while the focus in industry has emphasized economic issues and customer service agreements. Much academic fault management research has focused on one or more aspects of monitoring, analysis or adaption, omitting considerations of overall business objectives, budget constraints or the total cost of ownership. (There are a few exceptions, such as the International Workshop on Business-Driven IT Management, or BDIM, http://businessdrivenitmanagement.org/).

Research on monitoring solutions deal with the challenge of collecting system data with minimal impact on the target system's performance. Algorithms for failure prediction, automated diagnosis and root-cause analysis aim at figuring out whether a failure is looming or where a technical defect is located in multi-processor systems that deal with reconfiguring multiple machines consisting of diverse software and hardware platforms. Unfortunately, analyses of quantifiable effectiveness in terms of availability enhancement and cost reduction are rare.

The complexity of today's systems is driving a change of approach, even in fairly conservative sectors, such as safety-critical systems (Kurd et al. 2009), which tend to use static, non-adaptive practices, to steer runtime behaviour. We foresee the presence of self-management and adaptation as the norm rather than the exception in all dependable systems.As shown in Figure 1, we argue for a nested approach in order to obtain a more resilient, self-aware, and effective solution. This paper explores that idea and is structured as follows. We introduce our nested approach in Section 2 followed by six real-world examples that expose the inadequacies of current fault management practices in Section 3. The paper ends with conclusions and the call to arms.

2 Wheels within a wheel: a nested approach

Common fault management systems either implicitly or explicitly consist of four key steps that are often performed in a loop:

1. *Monitoring:* The managed system is monitored in order to collect data on the system's current status. The focus here is on minimizing the impact of monitoring on system performance while capturing enough important measures and events. Difficulties include acquiring the information needed to do effective fault detection, diagnosis, prediction, and recovery, while operating in heterogeneous as well as distributed environments.
2. *Analysis:* The monitoring data is analyzed for a variety of purposes: to characterize workload, to determine normal (fault-free) behavior, to evaluate whether and how well the managed system is meeting its performance/behavior expectations, and to determine possible culprits when the system deviates from those expectations. The outcome of analysis is one or more of the following: the detection of an existing fault, the localization of the fault, the prediction of an imminent failure, and the diagnosis of the underlying root cause(s) of the failure.
3. *Planning:* Once analysis determines the fault or the failure in the system, some action needs to be taken to address the problem. Typically, there are several ways to eliminate or alleviate it. A decision needs to be made to choose the countermeasure that is expected to be most effective. This step also includes determining the sequence of steps to execute, while ensuring that no undesired system state is entered during the transformation of the current state into the desired state. The measures for whether an action is "successful" include the likely effects of changes and the predictability of the system's state afterwards. Ultimately, good planning should derive the most effective recovery strategy to avoid a failure or minimize its damage.
4. *Execution:* The outcome of the planning step has to be executed. Challenges include the handling of legacy systems, distributed and replicated data, nondeterministic side-effects, multiple versions and operating environments, plans that are superseded by events that occur after they are made, configurability issues and the need for 24x7 operation.

This control loop is illustrated as the inner flow in Figure 1. A similar loop has been proposed by IBM in the context of autonomic computing (Jacob et al., 2005; Kephart & Chess, 2003), i.e., the Monitor, Analyze, Plan, Execute (MAPE) structure. Our focus is at a higher level: a holistic approach that optimizes for the overall goals by selecting, configuring, and combining an appropriate set of individual approaches and algorithms, together with their interactions.

It is not enough to make this selection statically, or for each component in isolation. Most fault management systems tend to address these steps fairly independently, rather than seek to understand and exploit the relationships among them. For instance, recovery actions in the planning step tend to be decided by what customers or end-users perceive to be acceptable or reasonable, under failures. On the other hand, instrumentation in the monitoring step often tends to be decided based on what is convenient (e.g., available off-the-shelf instrumentation packages) or cheap to monitor. Thus, it is rare to see either recovery-driven monitoring or a monitoring-driven approach. We argue for a fault management approach that is aware of the natural dependencies between the steps.

Rather than analyzing each step separately and optimizing for local goals, our emphasis is on thinking about the behavior of the management process itself, as indicated by the large dashed circle in Figure 1. More specifically, we are interested in understanding and managing the management system in order to:

- Align the management system with the business goals of fault management – not necessarily just the technical ones. In other words, from a system provider's perspective, fault management is only a means to meet business and technical objectives with minimal cost under business as well as technical objectives. Future fault management systems have to be aligned with these objectives; this means incorporating the likely costs and benefits of actions.
- Harmonize the interplay among the steps. For example, if monitoring certain variables is not improving analysis (as detected by variable selection methods, such as Liu & Yu (2005)) then perhaps they should no longer be monitored.
- Adapt the fault management system to system changes. As systems are subject to ongoing updates, upgrades and changing configurations, the fault management process needs to be adapted, too. For example, new monitoring variables might be selected that are useful for better analysis. In that case, monitoring as well as analysis should include this new data.

- Analyze the overall effect of a fault management system with respect to business and technical objectives. This includes a feasibility analysis and the probability of meeting overall objectives. Such an analysis might be accomplished by an estimation of expected costs and savings, and may encompass life cycle considerations as well as objectives such as energy efficiency.
- Determine the impact of each step on objectives. By analyzing the entire loop, the most critical steps can be identified. This helps to decide how much effort should be expended at each stage and guides how to eliminate potential overlaps (e.g., if planning is already determined by analysis).
- Last but not least, an analysis of the steps of fault management helps to design for fault manageability. This applies to both improving an existing system as well as the derivation of rules and patterns for how fault-manageable systems should be designed.

In addition to these overall goals, each step needs to be analyzed and adapted independently, as is indicated by the small dotted circles in the figure. For example, if failure prediction is performed in the analysis step, the prediction model's parameters need to be adapted to reflect changing behavior in the system (Salfner et. al., 2010).

In summary, we believe that the overall choice of what to do is a function of the costs and benefits of executing the four steps in a management system individually and together. One way is to use models that describe the management processes in the system, their effectiveness, and their behaviors. Such models are in addition to any models that the management processes may have of the underlying target system. Another approach is to communicate a global objective function to each of the processes, so they can adjust their behavior accordingly. (Note that this requires the ability to predict the emergent properties of the management system that results – itself a kind of model.) Regardless of how it is achieved, this kind of flexibility is likely to lead to a better result than adopting a rigid, static process with local optimizations. Ultimately, we also want the management system, and not just the target system, to be self-adapting.

To support our hypothesis we present six real-world examples from a broad range of application domains. The first one is explained in greater detail while the remaining five examples are only sketched for the sake of brevity.

3 Six real-world examples

The following six examples refer to applications of a variety of fault management methods that have been incorporated in a number of industrial systems ranging from data centers to embedded systems. They all demonstrate that the fault management problems are currently approached in a fragmented manner, and serve to establish that there is an urgent need for a holistic approach that takes into account both business and technical considerations, as we advocate in this paper.

3.1 Autopilot automated repair service

Autopilot (Isard, 2007) is an example of a data center management system that was designed with a specific objective: to keep the total cost of a data center, including operational and capital expenses, as low as possible. Autopilot is responsible for automating software provisioning and deployment, system monitoring, and carrying out repair actions. In the following paragraphs, we will focus on the automated repair component of the system.

Amongst the assumptions embedded in the design of Autopilot the two that are relevant for our purposes are: a) fault management processes are designed so that any process can be killed unexpectedly without destabilizing the system, and b) there is an order of magnitude difference in the costs of repair operations (reboot, re-image, and human intervention). An immediate consequence of these assumptions is that a good-enough plan simply (re-)applies the cheap solutions several times before resorting to the next-more-expensive one. This in turn establishes that there is very little need for expensive analysis, including a formal diagnostic process. Efficient and reliable detection of a possible problem is all that is needed: it does not need fault location for this purpose. Autopilot was originally designed

for managing search machines for the Windows Live search service, which are stateless and whose processes and algorithms fit the assumptions well.

Now, suppose that the "physical" hardware is a virtual machine. Re-imaging and rebooting are now much closer in cost: the image is likely to be shared (perhaps via copy-on-write techniques). Invoking people is probably a bad idea until the health of the supporting host has been determined. Furthermore, additional remedies might be available, such as migrating the virtual machine to a different host, an option with a comparable cost to re-imaging or rebooting. Now the choice of what to do is slightly less clear, and perhaps more effort should be expended in diagnosis to select an appropriate action. Similar considerations are in order for services involving storage, where both a reboot and a reimage may involve costly replications.

The point here is not that a different set of choices might be made, but that the set of choices is a function of the cost of the recovery mechanisms and the context in which they are being used. The system in charge of the overall execution of this process needs to take account of the costs, and benefits, of each of the stages as suggested in Section 2. Autopilot has evolved considerably and beyond the version in the work reported by Isard (2007), and has been adapted to several other properties. Some possible tools for evaluating the performance of an automated repair service and assessing whether the criteria in Section 2 are met, are presented by Goldszmidt et al. (2010).

3.2 Large-scale data scrubbing

File systems suffer from a very low rate of bit rot, that usually only becomes visible at large scale. The typical cause is latent software defects, triggered by rare combinations of partial failures, sometimes coupled to hardware glitches (Jiang 2008). To cope with this phenomenon, large-scale file systems implement a form of scrubbing, in which data is read and verified to be correct (e.g., by means of a checksum), or corrected (e.g., by overwriting it with a known correct copy) (Baker et al., 2006; Jiang et al., 2008).

How frequently should this scrubbing be performed? The answer depends on the cost of doing the reading/validation, the frequency of errors, and the cost of transmitting information to the place where decisions are made, or even the importance of the data stored. Different scrubbing techniques will have different costs, and different degrees of effectiveness; it is unlikely that one single technique will always be "best". (Imagine a decentralized file system spread across a continent: should the data checksum be sent to a central repository, or should a batch of checksums be collected and their checksum sent instead? The latter is cheaper to transmit, but generates more complexity if a defect is discovered.)

If errors are (relatively) common, the scrubbing rate should be increased to catch problem sooner, so remedial action can be applied, or the system should switch to more expensive, and more effective, error detection schemes. But if errors are rare, it may be appropriate to reduce the scrubbing rate, or use skimpier, lighter-weight techniques. Better yet would be to adjust the detection scheme to preferentially target those portions of the data that are especially at risk – perhaps because they are updated more often, or stored on hardware or software components with a higher-than usual defect rate.

Dynamically trading off scrubbing technique and frequency against effectiveness is an example of the kind of holistic approach we are discussing, and involves models of the effectiveness and costs of different scrubbing processes, as well as dynamically-built models of other parameters such as the underlying failure rates. What matters is the overall effectiveness of the control system, not just the costs of the individual processes in the management loop.

3.3 Cell phone credit checks

A common step to reduce fraud in telecommunication systems is to ensure that the caller has sufficient credit when setting up a new call. Under high load, this may not be possible and the call will not be handled, which is unsatisfactory to the customer. A different approach would be to let the call go through without the credit being checked under such conditions. The cost is the potential for an increased risk of fraud. Whether this is a good idea or not depends on the cost of doing the check, the

current load, the likelihood of fraud (or non-payment), the length of the call, and the amount of the potential loss (e.g., international calls might be checked more eagerly).

Implementation of such an approach requires thorough analysis and risk assessment. In order to be useful from a business perspective, such analysis needs to take global objectives as well as constraints into account. Additionally, in order to be able to perform the investigation, all steps need to be analyzed separately. That implies, for example, assessing the system's ability to correctly evaluate its current status, investigating the likelihood of making the right decision about whether to let the call go through or not, and understanding implications resulting from reconfiguring process handling (i.e., postponing the credit check). Without having a rather precise assessment of each part and an analysis of the interplay among the parts, the analysis cannot be trusted. And without a reliable analysis, no confident assessment of business risks and objectives can be achieved.

Mapping this problem on our “wheels within a wheel” methodology, we can consider fraud as a failure and aim at minimizing the cost of such failures while keeping the management cost as low as possible and maximizing the gain of uninterrupted phone traffic flow.

3.4 Storage system failure recovery

In this example, the goal is to produce a minimum-cost solution that meets a target availability or reliability goal; or (equivalently) the maximally available or resilient system for a given investment. Prior work (Keeton, 2004) has shown that it is possible to automate the decision-making at design time, using models of the underlying processes and their costs. The initial design process usually provides plenty of time to make decisions – such decisions are often taken on a timescale of days or weeks. However, this may not be true when a fault has occurred and a rapid response is needed, possibly with incomplete information. We still need a design for how to resolve the situation, but there may not be much time to come up with it. The design space may be smaller, but this is not guaranteed – it depends on the scope of the problem. Consider a data center disaster, which takes out a third of the available capacity, as one such case where a large-scale rethinking of resource allocations may be needed (Keeton, 2006).

The correct choice of what to do will depend on the urgency of the situation, the relative cost of making the decision against the penalties associated with delaying recovery, and the likelihood of those decisions being correct. For example, it may be better to apply a slower recovery sequence if it reduces the chance of incorrect inputs causing a bad outcome.

Again, we see an example of a need to adapt the design fault management processes dynamically, using a combination of situational information and models of each of the management processes and their behavior.

3.5 The (not quite) perfect deadlock detector

Assume there is a system that should perform 100,000 transactions per hour. Two deadlock detectors are available: one detects every deadlock but is expensive to run; the other costs less to run, but has less than 100% coverage. Which one should be used? The answer depends on the effectiveness of each detector (which might be a function of the offered load and contention), deadlock frequency, the cost of a transaction rollback, and the cost of recovery. Of course, determining these values has its own measurement and analysis costs. Again, we would like to make decisions using a model of the processes (the deadlock detectors and the measurement system), the load, deadlock frequency and resulting effects on the system as a whole. Even with two deadlock detectors, the problem can become complex quickly but a holistic approach may transform it into a manageable task, ending up with hybridized detectors, each of them activated based on, e.g., load and contention.

3.6 Embedded system diagnosis and upgrades

Modern trucks have a complex control structure in their engine management system (EMS) to implement efficient monitoring of (legal) environmental requirements, support for service mechanics, and enforcing safety properties. The EMS software controls approximately 100 components, including sensors, actuators, and even virtual sensors that compute physical quantities based on a model. EMS

operates in harsh conditions and is subject to transient and permanent faults. To get an idea of the complexity, the fault management in a Scania EMS involves setting one of three degradation modes for each of the components involved, based on signals sent from the components and 400 diagnostic tests (Johansson, 2007).

Fault management aims at finding a balanced trade-off between safety violation risks, and the cost of immediate maintenance (mainly caused by taking the vehicle off the road). Currently the trade-off is based on predefined rules for diagnosis and run-time adaptation in each member of the product family. Since there are many variations in the product line, the trade-off has to be defined manually for every combination of components. The upgrade problem is therefore to produce a new system based on some components from the earlier models, and to demonstrate that technical as well as safety requirements are met in a way that overall business objectives are optimized. Having a model to incorporate all system variations and a holistic approach considering global objectives can improve the design process significantly. This could also incorporate constraints derived from the usage profile and history of faults.

In addition, such a holistic approach may lay the foundation for a self-managing EMS that optimizes decisions at runtime on the road. An effective adaptive system reflects on the adaptation of the diagnosis subsystem at the same time that any new functions are added to the functional capabilities in new version of the product. During the design phase, one ought to seek a balance between successful interventions in the run-time behavior of the system in presence of faults, on a per truck basis, and the costs that arise to derive future versions of the product.

4 Conclusions

The current state-of-the-art is that most practitioners in the autonomic, self-* area talk and write about how their systems accomplish some goal. While this is good, we believe that there is more to do. In particular, by making management systems themselves be the target of analysis and adaption, it will be possible to achieve a much greater range of beneficial behaviors. This increased range will be more business- or mission-relevant because it will be able to take a much broader view of ”goodness” that is aligned with overall business and technical objectives. Taking a holistic view is the key here. Optimizing each fault management process is not just insufficient – it may even be the wrong thing to do. The best, most resilient, diagnosis solution or failure prediction algorithm may be unnecessary if recovery is lightweight and quick, and exploration of alternative recovery actions are cheap and easy. In turn, this will raise the bar for deciding whether a fault management system is providing benefit. We believe it should not be sufficient to provide a couple of worked examples of a management or control system “doing the right thing”. The interesting questions need to include “Compared to what?”, “How robust is that to changing assumptions?”, “How effective are the solutions?”, and “What is the return on investment (how much does the new fault management cost versus how much it saves)?”

We commend this line of reasoning to all who are investigating self-managing systems, and look forward to designs and assessments of such management systems in the next years.

Acknowledgements

This work has emerged as a result of comparative analysis of problems presented and discussed in Schloss Dagstuhl Seminar 09201 on Self-healing and Self-adapting Systems in May 2009. The authors gratefully acknowledge the organizers of the workshop and the cooperation-promoting ambience at Dagstuhl.

References

Baker, M., Shah, M., Rosenthal, D.S.H., Roussopoulos, M., Maniatis, P., Giuli, T.J., & Bungale, P. (2006). A fresh look at the reliability of long-term digital storage. *1st SIGOPS/EuroSys European Conference on Computer Systems* (pp. 221-234). ACM.

Goldszmidt, M., Budiu, M., Zhang, Y., & Pechuk, M. (2010). Toward automatic policy refinement in repair services for large distributed systems. *LADIS workshop* (Oct. 2009). Published as *SIGOPS Operating Systems Review*, 44(2), 47-51. ACM.

Isard, M. (2007). Autopilot: automatic data center management. *SIGOPS Operating Systems Review*, 41(2), 60-67. ACM.

Jacob, B. et al. (2005). On Demand Operating Environment: Managing the Infrastructure (Virtualization Engine Update). IBM Redbooks, 2nd edition. IBM.

Jiang, W., Hu, C., Zhou, Y., & Kanevsky, A. (2008). Are disks the dominant contributor for storage failures? A comprehensive study of storage subsystem failure characteristics, *ACM Transactions on Storage*, 4(3), Article 7.

Johansson, F. (2007). Fault handling architecture in embedded automotive systems. Master's thesis. LITH-IDA-EX-07/065-SE, Dept. of Computer & Information Science, Linköping University, Sweden.

Keeton, K., Beyer, D., Brau, E., Merchant, A., Santos, C., & Zhang, A. (2006). On the road to recovery: restoring data after disasters. *1st SIGOPS/EuroSys European Conference on Computer Systems* (pp. 235-248). ACM.

Keeton, K., Santos, C., Beyer, D., Chase, J., & Wilkes, J. (2004). Designing for Disasters. *3rd Conference on File and Storage Technologies (FAST'04)* (pp. 59-62). USENIX.

Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing, *IEEE Computer*, 36(1), 41-50.

Kurd, Z., Kelly, T., McDermid, J., Calinescu, R., & Kwiatkowska, M. (2009). Establishing a framework for dynamic risk management in 'intelligent' aero-engine control. In Buth, B., Rabe, G., & Seyfarth, T., (Eds.), *Computer Safety, Reliability, and Security*, Volume 5775 of *Lecture Notes in Computer Science*, Chapter 26, (pp. 326-341). Springer, Berlin/Heidelberg.

Liu, H. & Yu, L. (2005). Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, 17(4), 491-502.

Salfner, F., Lenk, M., & Malek, M. (2010). A Survey of Online Failure Prediction Methods. *ACM Computing Surveys*, 42(3), Article 10.

Indexing terms: fault management, recovery, monitoring, business objectives, failure prediction, adaptation, self-* , diagnosis, upgrades, cost savings