Wheels within Wheels: Making Fault Management Cost-Effective

Moises Goldszmidt, Miroslaw Malek, Simin Nadjm-Tehrani, Priya Narasimhan, Felix Salfner, Paul A.S. Ward, and John Wilkes*

Working Group 7
Dagstuhl Seminar 09201
Self-Healing and Self-Adapting Systems
Dagstuhl, Germany
May 10–15, 2009

Abstract. Local design and optimization of the components of a fault-management system results in sub-optimal decisions. This means that the target system will likely not meet its objectives (under-performs) or cost too much if conditions, objectives, or constraints change. We can fix this by applying a nested, management system for the fault-management system itself. We believe that doing so will produce a more resilient, self-aware, system that can operate more effectively across a wider range of conditions, and provide better behavior at closer to optimal cost.[1]

1 Motivation

We start with an automated repair system similar to the repair service of Autopilot [2]. The unit of failure in Autopilot is a "device" rather than a process. The available remedies range from doing nothing, automated reboots and re-imaging, up to several levels of expert human intervention. Faults are detected using monitoring agents that send event signals to the repair service. The life-cycle of each device is managed by a state machine; the state changes as a function of the signals from the agents, the repair actions, and the (recent) history of the device. The automated repair policy is defined to be a mapping from the state, the signals, and the history to a repair action. The repair service in Autopilot was designed to handle distributed systems on the order of hundreds of thousands of machines.

In the very original version of the service, the optimal policy was clear because there was an order-of-magnitude difference in cost for each stage in this escalation process: a system could be rebooted several times in the time it would take to re-image it; a system

* Alphabetic ordering to protect the guilty.

[1] Note: like all working documents, not all members of the Working Group 7 team agree with all of the contents of this paper! We all agreed, however, that looking at and optimizing each component in isolation in the loop will not only lead to suboptimal solutions and unnecessary complications, but may lead the whole system astray. The point being that a holistic approach, guided by a top-down objective function, should dictate the design and interactions of each component.

Dagstuhl Seminar Proceedings 09201
Combinatorial Scientific Computing
http://drops.dagstuhl.de/opus/volltexte/2009/2102

could be re-imaged for much less cost than a single visit by a person. This naturally led to an aggressive attempt to (re)apply the cheap solutions several times before resorting to the next-more-expensive one.

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. Further, 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. Certainly, the most aggressive option could be taken, or perhaps more effort could be expended in diagnosis before doing anything, in the hope that this might be less disruptive (*e.g.*, to locally saved state).

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. 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.

We believe this pattern is a common one: the overall choice of what to do is a function of the costs and benefits of executing the various steps in a management system. It is better to take these costs and benefits into account when deciding what to do, compared to a static design. One way to do this is to use meta models, which describe the management components in the system, their effectiveness, and their behaviors — *e.g.*, are they disruptive? This is above and beyond any models that the management components may have of the underlying target system. There are others: for example, a global optimization function could be communicated to each of the components, and they could adjust their behavior to take that into account.

Regardless of how it is achieved, being flexible in this way is likely to lead to a better result than adopting a rigid process. We want the management system to be self-adapting, too — not just the target system.

2 Other Examples

As suggested above, we believe this is a fairly common pattern in management systems. The following examples attempt to make this case.

Large-Scale Data Scrubbing

Very big file systems frequently suffer from bit rot: asynchronous data loss. To cope with this, many implement a form of scrubbing — data is read and verified correct (*e.g.*, by means of a checksum), or corrected (*e.g.*, by overwriting it with a known correct copy) [1, 3]. How aggressive should this scrubbing be? The answer depends on the cost of doing the reading/validation, the frequency of errors, and, perhaps, the cost of transmitting information to the place where decisions are made. (Imagine comparing an MD5 checksum against a known good value: should the checksum be sent to a central repository, or should a batch of checksums be collected and their checksum sent instead?) If errors start to become more common, the frequency probably ought

to be increased, to reduce the MTTR (mean time to repair), and thus improve the data reliability. The underlying target system is the file system; the management system is the automated data-scrubbing and repair one; the decision of where to put effort requires meta-models of the scrubbing and comparison components, and the underlying failure rates.

Cell Phone Credit Checks

It is common to ensure, when setting up a new call, that the caller has sufficient credit, in order to reduce fraud. Under high load, this may not be possible, so the system will let a call go through without the credit check in order not to lose it. Nevertheless, the caller might be charged for that call later. Whether it is a good idea is a function of the relative cost of doing the check or not, 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). A cost-effective solution has to take all these factors into account.

Storage-System Failure Tolerance

Here, 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 [4, 5] has shown that it is possible to automate the decision-making at design time, using models of the underlying components and their costs. The design process usually provides plenty of time to make decisions. However, this may not be true when a fault has occurred and a rapid response is needed, possibly with incomplete information. The correct choice of what to do will depend on the urgency of the situation, and the relative cost of making the decision against the recovery time, 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. That is, we need to adapt the design process, using information about each of the management components and their behavior.

The (Not Quite) Perfect Deadlock Detector

Assume there is a system that should perform 100,000 transaction 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 to use? The answer depends on the effectiveness of each detector (which might be a function of the offered load and contention), 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 components (the deadlock detectors and the measurement system), the load, and resulting effects on the system as a whole.

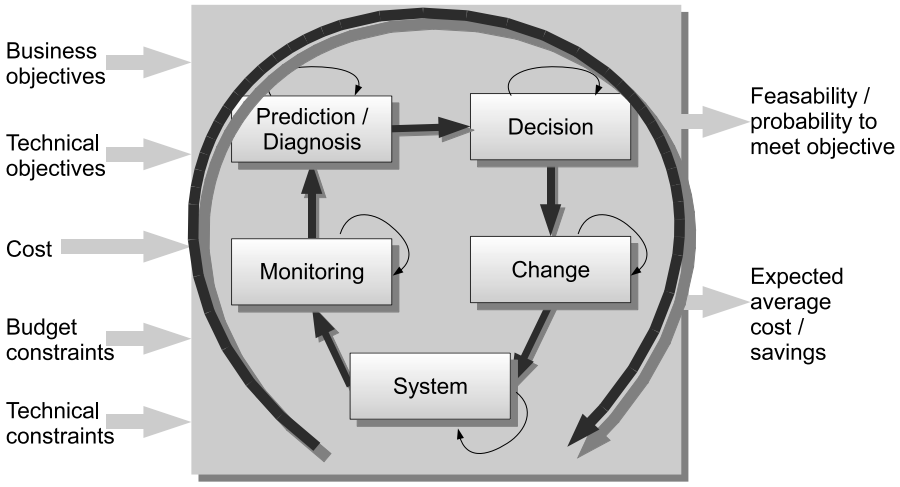

Fig. 1. Wheels within Wheels

3 A Meta-Model for Management Systems

We found it helpful to identify commonly occurring elements in the control/management systems that we were using to motivate this thinking. Fig. 1 illustrates what we came up with. It largely follows the MAPE (Monitor, Analyze, Plan, Execute) approach familiar to followers of autonomic computing. The difference here is that the emphasis is on thinking about the behavior of the management process itself, not the target system. That is, we are interested in managing the management system itself, which can determine, for example, how much effort should be expended at each stage.

The exterior of the figure illustrates a few of the environment attributes that the meta-management system has to take into account. The main loop represents the normal operation of the underlying management system, as well as the need to model each of the components and how well they perform. We need to be concerned with the following aspects:

1. Monitoring: how well is the system doing? What is it doing? Is it doing it in a timely fashion?
2. Analysis and diagnosis: is the managed system doing what it is expected to do? How well is it doing it?
3. Prediction: Will the system be running well in the near future? What would happen if we changed, *e.g.*, a configuration parameter, the set of components used, or the overall system configuration?
4. Decision and planning: given the observed behavior, a desired behavior, and predictions about the likely effects of changes, what is the appropriate management-system setup that should be used (this is basically exploring the space of potential designs and their consequences, using the predictive models)?
5. Planning: if a change is needed, how do we get from the current state to the new one? In some situations this is simple: the management system can be disabled

while a switch is made, while the underlying target system continues operation. In other cases, this is much harder (*e.g.*, where active control is needed to avoid disaster, or where there is state that is incompatible between the old and new configurations). In some cases a detailed plan is appropriate; in others this will rapidly get out of date, and so an incremental approach is required.

6. Change: given a desired end state, a current state, and (perhaps) a plan, go ahead and make the change

4 Research Agenda and Call to Arms

The current state-of-the-art is that most practitioners in the autonomic, self-* space talk and write about how their systems accomplish some goal. While this is good, we believe that there is more to do. In particular, if we make those management systems themselves to be the target of a management system, we believe that it will be possible to achieve a much greater range of behaviors. Our hypothesis is that this increased range will be more business- or mission-relevant, at least in part because it will be forced to take a much broader view of "goodness" than do the lower-level management components.

Taking a holistic view is the key here. Optimizing each component is not just insufficient — it may be the wrong thing to do:

- The best, most resilient, diagnosis solution may be unnecessary if recovery is lightweight and quick, and exploration of alternative problems cheap and easy.
- A really cheap design process may be useless if it the cost of a change outweighs any modeling and prediction efforts.
- A planning step may produce terrible results if it is not fed reliable, high-quality inputs, and it is worth doing so, even if gathering them costs more.

In turn, this will raise the bar for deciding whether a 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?" and "how robust is that to changing assumptions?"

We commend this line of reasoning to all who are investigating self-managing systems, and look forward to assessments of such meta-management systems in the literature in the next year or two.

References

1. Mary Baker, Mehul Shah, David S. H. Rosenthal, Mema Roussopoulos, Petros Maniatis, TJ Giuli, and Prashanth Bungale. A fresh look at the reliability of long-term digital storage. *SIGOPS Oper. Syst. Rev.*, 40(4):221–234, 2006.
2. Michael Isard. Autopilot: automatic data center management. *SIGOPS Oper. Syst. Rev.*, 41(2):60–67, 2007.
3. Weihang Jiang, Chongfeng Hu, Yuanyuan Zhou, and Arkady Kanevsky. Are disks the dominant contributor for storage failures?: A comprehensive study of storage subsystem failure characteristics. *Trans. Storage*, 4(3):1–25, 2008.

4. Kimberley Keeton, Cipriano Santos, Dirk Beyer, Jeffrey Chase, and John Wilkes. Designing for disasters. In *FAST '04: Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, pages 59–62, Berkeley, CA, USA, 2004. USENIX Association.
5. Kimberly Keeton, Dirk Beyer, Ernesto Brau, Arif Merchant, Cipriano Santos, and Alex Zhang. On the road to recovery: restoring data after disasters. In *EuroSys '06: Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, pages 235–248, New York, NY, USA, 2006. ACM.