Event-Based Self-Management

Paul A.S. Ward[*]*, Dwight S. Bedassé, Tao Huang, Mohammad A. Munawar, and Jai Jun Wu*
Department of Electrical and Computer Engineering, University of Waterloo
200 University Ave. W.
Waterloo, ON, Canada
{*pasward,dsbedass,t6huang,mamunawa,jjwu*}*@ccng.uwaterloo.ca*

Abstract

The behaviour of distributed applications, services, and systems can be modeled as the occurrence of events and their interrelationship. Event data collected according to the event model is stored within a partial-order data structure, where it can be reasoned about, enabling autonomic feedback control. This paper summarizes a decade's worth of research on distributed-system monitoring and control, and shows how that work is directly applicable to the problem of self-managed distributed systems. We will identify the major issues and open problems in self-management, and discuss the various approaches we are taking to addressing those problems.

Keywords

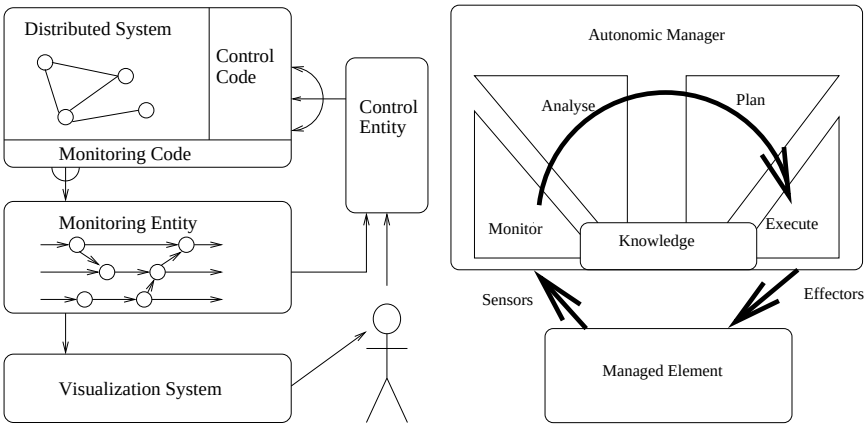
Autonomic computing, program steering, self management.

1. Introduction

The architecture of tools for monitoring and controlling distributed applications, services, and systems is broadly similar, and can be characterized as shown in Fig.1(a). A variety of such tools have been built over the years, including ATEMPT [4, 5], Object-Level Trace [2], POET [6], POTA [9], and Log and Trace Analyzer [1]. The distributed system is instrumented with monitoring code that captures significant event data. Ideally, the information collected will include the event's process and thread identifiers, number, and type, as well as partner-event identification, if any. This event data is forwarded from each process to a central monitoring entity which, using this information, incrementally builds and maintains a data structure of the partial order of events that form the computation [7]. That data structure may be queried by a variety of systems, the most common being visualization engines for debugging and control entities for program steering.

The intent of this paper is first to observe the similarities and differences between this traditional distributed-system observation and control architecture, and the recent "MAPE"-loop architecture of IBM [3]. Having compared the approaches, we will identify what issues would have to be addressed to apply that traditional approach to the problem of autonomic management of distributed systems. We then discuss the various methods we are applying in addressing these problems. We close by summarizing our view of the hard problems in this area.

[*]This work was supported in part by IBM Autonomic Computing



(a) Traditional Distributed-Systems Management Architecture

(b) MAPE-Loop Architecture

2. Mapping Distributed-System Management to the MAPE Loop

The MAPE-loop architecture, developed by IBM [3], is shown in Fig.1(b). While superficially different from traditional distributed-systems management, it is not difficult to map from the one architecture to the other. The mapping is as follows:

Managed Element : Sequential entity
Sensors : Monitoring Code
Monitor : Monitoring Entity
Analyze : Visualization System/Person
Plan : Person/Control Entity
Execute : Control Entity
Knowledge : Person/Control Entity
Effectors : Control Code

Having defined the mapping, we now look at these components, and observe in what ways they are similar, and in what ways they differ.

First note that the managed element within the MAPE model corresponds to a sequential entity within the distributed-systems management model. This mapping is one-to-one, insofar as the concerns of both approaches is similar. For example, a Java bean might be a managed element in the MAPE approach, and a sequential entity in distributed-systems management approach. However, a critical difference between the two approaches is that in an AC environment, the managed entity may well be an encapsulated component. That is, it may not be instrumented in the same manner as its calling entity.

This leads to our second observation, which is that the monitoring code in a traditional distributed-system management tool will typically have been added explicitly for the purpose of management. It may have been added manually, or by automated code insertion, but it will be designed for the management purpose. For example, in the debugging context, the relevant compiler option will add the necessary code. In autonomic systems, by

contrast, we cannot presume that the system will be instrumented expressly for the purpose of management. While this is the ideal, it is not plausible to presume that various enterprise software vendors will, in the near future, agree on a single, all-encompassing format for data collection (notwithstanding the efforts in that direction of approaches such as the Common-Base Event (CBE) format [8]). In particular, while in the introduction we observed what the event data should ideally contain, we note that the CBE format does not satisfy this requirement, nor can we expect legacy systems to be retrofitted with these desiderata. What this means in practice is that the monitoring entity of a distributed-system management tool must be reworked to recognize the limitations of the input event data. In this regard, the Log and Trace Analyzer [1] has done the most-significant work. It scavenges data from log files of large enterprise systems, and then provides options for known correlation, heuristic correlation, and a generic plugin framework, allowing for arbitrary matching of events as required.

Similarly, effectors may not be exactly as desired to the degree that control code is within an instrumented system. Rather, exiting input mechanisms will have to be adapted to the needs of the control entity.

Given that the sensor and effector data is likely quite different in the MAPE model than in traditional distributed-systems management, it might appear that there is little value in trying to adapt the traditional approach to this new environment. However, observation of the Log and Trace Analyzer project suggests that wrapping legacy systems and creating plugins can be effective in creating the illusion of a system that has been purposefully instrumented for management.

This leaves the remaining elements. With effective sensor/effector wrapping, most of the existing body of work in the area can then be applied to autonomic computing, though observing that the intent is to remove the human from the loop in Fig.1(a). We therefore turn to the question of what the open problems are within the area, as seen from the perspective of distributed-systems management, and our approaches to those problems.

3. Open Problems and Our Work

Given the mapping we have seen between the two areas, our research work has focused on bringing research from distributed systems management into the autonomic control of distributed systems. In this, we have focused on four main projects. First, we have begun to address problems of scale. Specifically, we are attempting to enable distributed systems management over orders-of-magnitude more entities than has been the case. While this work has thus far been in the area of scalable data structures [11], it has begun to focus more recently on the problem of monitoring cost. Specifically, as the systems become larger, they produce significantly more data. Preprocessing of that data is essential to enable scalable management. Second, we are addressing the problem of application diversity through the use of event correlators. By flowing correlators through the monitoring process it is possible to deterministically associate events with tasks [12]. This enables reasoning over task behaviour, substantially improving problem determination capabilities. Third, we have started to look at predicate detection performance [10]. Efficient pattern detection will be crucial for autonomic control.

Finally, and most recently, we have started to address the problem of when, where and how much data to gather. The questions we are attempting to answer are first: how much monitoring is required to determine if more monitoring is required. That is, what is the least monitoring necessary to allow us to consider the possibility of a problem. Second, given that we discovered we needed more monitoring information, have we collected enough to resolve the problem, or did we miss relevant data.

4. Conclusions

In this paper we have summarized prior, very briefly, research on distributed-system monitoring and control, and shown how that work is applicable to the problem of self-managed distributed systems. We have identified some of the major issues and open problems in self-management, and discuss the various approaches we are taking to addressing those problems.

References

[1] IBM Corporation. Log and trace analyzer for autonomic computing. Online documentation available at: http://www.alphaworks.ibm.com/tech/logandtrace.

[2] IBM Corporation. Object level trace. Online documentation available at: http://www-106.-ibm.com/developerworks/websphere/WASInfoCenter/infocenter/olt_content/olt/index.htm.

[3] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41– 50, 2003.

[4] Deiter Kranzlmüller, Siegfried Grabner, R. Schall, and Jens Volkert. ATEMPT — A Tool for Event ManiPulaTion. Technical report, Institute for Computer Science, Johannes Kepler University Linz, May 1995.

[5] Dieter Kranzlmüller. *Event Graph Analysis for Debugging Massively Parallel Programs*. PhD thesis, GUP Linz, Linz, Austria, 2000.

[6] Thomas Kunz, James P. Black, David J. Taylor, and Twan Basten. POET: Target-system independent visualisations of complex distributed-application executions. *The Computer Journal*, 40(8):499–512, 1997.

[7] Leslie Lamport. Time, clocks and the ordering of events in distributed systems. *Communications of the ACM*, 21(7):558–565, 1978.

[8] David Ogle, Heather Kreger, Abdi Salahshour, Jason Cornpropst, Eric Labadie, Mandy Chessell, Bill Horm, and John Gerken. Canonical situation data format: The common base event. Online documentation available at: http://xml.coverpages.org/IBMCommonBaseEventV111.pdf.

[9] Alper Sen and Vijay K. Garg. Partial order trace analyzer (POTA) for distributed programs. In *Proc. Workshop on Runtime Verification*, 2003.

[10] Paul A.S. Ward and Dwight S. Bedassé. Fast convex closure for effi cient predicate detection. Submitted to EuroPar 2005 ; available at http://www.ccng.uwaterloo.ca/
- pasward/europar2005.pdf.

[11] Paul A.S. Ward, Tao Huang, and David J. Taylor. Clustering strategies for cluster timestamps. In Rudolf Eigenmann, editor, *Proceedings of the 2004 International Conference on Parallel Processing*, pages 73–81. IEEE Computer Society, 2004.

[12] Paul A.S. Ward, Jiajun Wu, and David J. Taylor. Collecting transaction data in event-monitoring tools. In preparation.