AIMS: Robustness Through Sensible Introspection

Fabián E. Bustamante, Christian Poellabauer and Karsten Schwan
College of Computing, Georgia Institute of Technology
Atlanta, Georgia 30332
{fabianb, chris, schwan}@cc.gatech.edu

1 Introduction

As individuals and organizations increasingly rely on services provided by complex computing systems for their everyday tasks, system dependability becomes a main concern. The dependability of a system is commonly defined as the property of it such that reliance can justifiably be placed on the service it delivers [1] and includes, as special cases, attributes such as reliability, availability, safety, and security. In addition, dependable systems must also be robust when facing the "messiness" of the real world, delivering correct service across a wide range of operational conditions and failing gracefully outside that range [15, 16].

A common approach to obtain robust dependable systems is trying to predict the system's future operational conditions and provision it accordingly. This approach, however, normally translates on high costs due to over-provisioning and costly system updates (for its web site, Charles Schwab maintains capacity at three to five times peak volumes [22]), and when predictions fail it can result on high average down-time with its associated costs [1], estimated at about $\$8,000$ per hour [5, 25].

In response, the systems community has identified as an important focus for research the design of computing systems capable of adapting to predictable changing environments [2, 11, 12, 20] and, in this context, introspection has proven to be a useful approach [3, 4, 13, 17, 18, 23]. Introspection is the ability to continuously monitor system behavior and adapt to changing conditions. Central to the process of providing introspection is the collection, aggregation, and processing of monitoring data. Probes are inserted in different parts of the systems to collect raw monitoring data about current hosts' hardware capabilities, dynamically compute statistics on environmental or systems conditions, or collate information on systems requests. The collected data is then selected and integrated to produce system-specific metrics, diagnose potential problems, and select among different adaptive measures to enact.

For robustness, however, the introspective component itself needs to be dynamically adaptive: the same complexity that initially drove us to adopt introspection along with the "messiness" of the real world mean that the parts monitored, the monitoring granularity and even the processing done on the collected data are bound to change dynamically. Since *(i)* it is effectively impossible to predict all information needed for introspection, *(ii)* even if we try, no introspective system will be able to manage the amount of data necessary to select the right adaptation to an overwhelming number of possible system conditions, for the introspective component to be useful in the real world it must be dynamically adaptive, and *(iii)* the "right" adaptation may be situation dependent as well.

Our work at Georgia Tech focuses on exploring the idea of dynamically adaptive introspective components for future systems. To this end, we are building *AIMS*, an *Adaptive Introspective Management System* through which monitoring probes (or *agents*) can be (un-)installed at runtime, their execution can be finely tuned dynamically, and the processing done on the collected data can be changed as needed.

In the reminder of this paper, we describe the proposed architecture, present some details on our current implementation, and describe our initial steps for the application of our ideas on a public *ftp* service.

2 Architecture

AIMS offers a push-based/active customizable service for environment- and self-monitoring. Hosts of interest to an application must become AIMS nodes. Different objects (including devices) at AIMS nodes are the sources of monitoring data. AIMS nodes connect to other nodes in specific sets or AIMS networks. AIMS clients connect to AIMS networks through an AIMS node that could reside on its own or another host. Clients express their interests in different monitored objects by requesting state reports on such objects or selectively registering themselves with the moni-

[1] Delta Air Lines experiences recovery delays of up to several hours in response to partial system outages [10] and EBay's 22-hour crash in June 1999 cost the company more than $\$5$ million in returned auction fees [25].

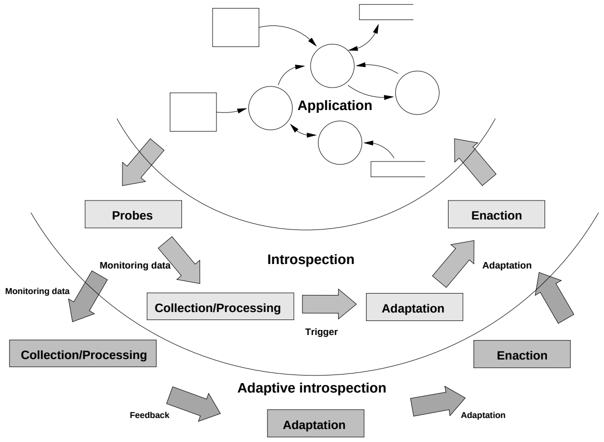

Figure 1. AIMS adaptive introspection.

toring streams associated with those same objects. Requests for continuous reports follow a *lease* model: they remain active for a user-specified (but limited) time span after which they are cancel. Previously issued requests can be re-issued or canceled, before lease expiration, using their request IDs.

Clients can select a subset of the data made available by different sensors, and integrate that data to produce application-specific monitoring metrics and decide on possible adaptations. In addition, filters in AIMS can be instantiated directly in the source of monitoring data streams with the corresponding savings in bandwidth, and in sink or source processing.

Sensors maintain histories of measurements. In order to forecast performance availability, AIMS includes a library with a number of predictive algorithms including mean-based methods such as running, trimmed, and sliding window averages as well as minimum, maximum, median and support to build autoregressive models.

An application using AIMS can easily redefine, at runtime, its monitoring views by adapting/replacing the filters used for monitoring as its workload and environment change. In this manner, applications have an additional level of adaptation that would not be possible were the monitoring and adaptation mechanisms coded statically for each of them.

Figure 1 depicts an adaptation hierarchy: distributed system resources are monitored and adapted (or controlled) by the first-level adaptation mechanisms. At the second level, the effectiveness of this adaptation is evaluated, and the first-level monitors and controllers are adapted, if required.

As the operational environment of an application changes and the administrator's understanding evolves, the focuses of monitoring and possible or useful adaptations change as well. Researchers have developed a multiplicity of tools for system instrumentation even helping developers and users select suitable instrumentation points. There is, however, the danger of over/under instrumentation. Overly aggressive instrumentation results in potentially overloaded and therefore, non-robust systems. Too conservative instrumentation may results in insufficient information to make sound adaptation decisions. On the other hand, wrong adaptation decisions (perhaps due to insufficient or "old" monitoring information) can be non-effective or directly counterproductive [13]. In sum, the monitoring, data processing, and adaptation part of the introspective component need to be dynamically adaptable. The monitoring data filters/analyses and the adaptation tasks performed based on them need to be (re-)deployed dynamically and executed efficiently; there is a need for light-weight ways to adapt sensors functionality, data processing and the adaptation routines themselves. The goal of adapting the introspection system is to improve the component's efficiency for a given execution instance: tuning the granularity of sensors, selecting the "right" prediction utility for the given monitoring stream's characteristics, reducing the monitoring overheads and perturbations [9, 24], choosing the "right" adaptation at a given point in time, and dealing with unpredictable disasters [17]. Clearly, all this needs to be done without making the adaptive introspective component so heavyweight as to deny its benefits.

3 Implementation Details

Currently, AIMS nodes report information on memory availability, CPU load, disk free space, up-time, and number of users currently logged on, as well as host name, IP, number of CPUs, and configure triplet (as reported by GNU config.guess). Status reports of network paths between two AIMS nodes include latency and bandwidth. AIMS uses a combination of passive and active sensors as needed [26]. Passive sensors exercise an external system utility, such as `uptime`, and scan the utility's output to obtain the required information. Active sensors, on the other hand, must conduct a performance experiment, such as timing a message exchange between two hosts, to measure the availability of the monitored resources.

A recent result of our work includes dynamically insertable kernel agents, which monitor system resources. The feature of this mechanism distinguishing it from past work is its ability to customize the monitors according to a client's specific needs. Applications manipulate these agents via an interface implemented as an extension to the Linux virtual file system `/proc` [14].

For adaptation, our approach includes the adaptation of kernel-level resources. While it is straightforward to inspect and manipulate system resources within the kernel, it takes costly interfaces to present to user-level manager (often repeated system calls are needed to determine sys-

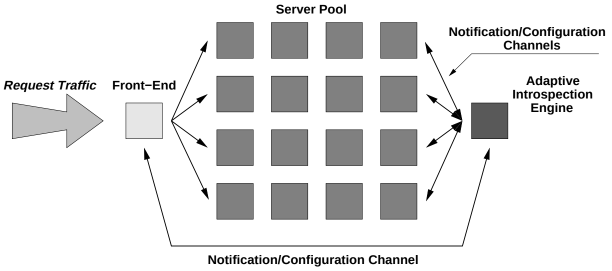
Server Pool
Notification/Configuration Channels
Request Traffic
Front−End
Adaptive Introspection Engine
Notification/Configuration Channel

Figure 2. Cluster-based service with adaptive introspection.

tem information or adapt system resources). In addition, we have found that fine-grain kernel-level management of resources can provide applications with benefits not derived from coarser-grain, user-level management [13].

Resource controllers modify resource allocations to a specific application based on the requirements of the application and the availability of resources. Resource controllers use information from monitoring agents for their decision-making.

We have built a kernel-based event service aimed at supporting the coordination among multiple kernel services in distributed systems [19]. Event and publish/subscribe services have become prevalent in distributed applications ranging from virtual reality, avionics, to support for mobile users [8, 21, 6]. Our current work investigates the utility of a kernel-based event service for resource management, distributed monitoring, and load balancing mechanisms.

4 An Example in Cluster-based Servers

We have started to apply this work to the monitoring of cluster-based Internet services. In particular, we are working on an *ftp* service for publicly available software provided by the CERCS research center in the College of Computing at Georgia Tech. The high-level architecture of the adaptive Internet service (Figure 2) is similar to the one described by Chase et al. [7]. As with their work, the goals are to maximize server performance and minimize the cost (energy) by concentrating requests on a minimal set of servers. That is, some servers run at near 100% utilization, while others are idle.

A cluster of servers consists of a front-end node and a number of back-end nodes, with incoming requests being forwarded by the front-end to one of the back-end nodes. The individual cluster machines monitor their loads (CPU utilization, queue lengths, etc.) and exchange such information with each other via a kernel-level event channel. This information is used (i) as a basis for admission control and (ii) for load balancing. Further, this information is also collected by a centralized **adaptive introspection engine**, which is able to adapt the front-end (e.g., to concentrate requests or to react to failing backends) and the back-ends (e.g., to vary queue lengths, change resource allocations, or modify forwarding algorithms). This forms the first level of the adaptive introspection mechanism. The second level is formed by monitors at the hosts measuring the effectiveness of the monitors and adaptation mechanisms. Again, this information is collected at the central adaptive introspection engine via another event channel. The engine is able to modify the behavior of the monitors (e.g., frequency of monitoring), the behavior of the event service (e.g., filters aggregate events or block events from being sent to idle machines), and the controllers (e.g., change thresholds or algorithms).

5 Conclusion

Our society increasingly relies on dependable complex computing systems. To be useful, dependable systems must also be robust when facing unpredictable changes to their operating environments. Introspection has proven to be a helpful approach in the design of dynamically adaptable computing systems. We argue that, for robustness, the introspective component itself needs to be dynamically adaptive since *(i)* it is effectively impossible to predict all information needed for introspection, *(ii)* even if we try, no introspective system will be able to manage the amount of data necessary to select the right adaptation to an overwhelming number of possible system conditions, and *(iii)* the “right” adaptation may be situation dependent as well.

At Georgia Tech we are exploring the idea of dynamically adaptive introspective components for future systems. To this end, we are building *AIMS*, an *Adaptive Introspective Management System* through which monitoring probes (or *agents*) can be (un-)installed at runtime, their execution can be finely tuned dynamically, and the processing done on the collected data can be changed as needed.

We are exploring our ideas on the context of a cluster-based *ftp* service, consisting of a front-end and a number of back-ends and a centralized adaptive introspection engine. Our future work will use adaptive mechanisms in the front-end and in the back-ends to adjust admission control and load balancing mechanism. AIMS will adapt these adaptation mechanisms to maximize the performance and robustness of the introspective system.

References

[1] Mario Barbacci, Mark H. Klein, Thomas H. Longstaff, and Charles B. Weinstock. Quality attributes. Technical Report

CMU/SEI-95-TR-021, Carnegie Mellon University, Pittsburgh, PA, 1995.

[2] Thomas Bihari and Karsten Schwan. Dynamic adaption of real-time software. *ACM Transactions on Computer Systems*, May 1991.

[3] Aaron Brown, David Oppenheimer, Kimberly Keeton, Randi Thomas, John Kubiatowicz, and David A. Patterson. Istore: Introspective storage for data-intensive network services, March 1999.

[4] Fabián E. Bustamante, Greg Eisenhauer, Patrick Widener, Karsten Schwan, and Calton Pu. Active streams: An approach to adaptive distributed systems, May 2001.

[5] Bruce Caldwell. Cost of downtime. *Information Week*, May 12 1997.

[6] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, August 2001.

[7] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, and Amin M. Vahdat. Managing energy and server resources in hosting centers, October 2001.

[8] Greg Eisenhauer, Fabian E. Bustamante, and Karsten Schwan. Event services for high performance computing. In *Proc. of the 9th High Performance Distributed Computing (HPDC-9)*, Pittsburgh, PA, August 2000.

[9] Greg Eisenhauer, Weiming Gu, Karsten Schwan, and Niru Mallavarupu. Falcon - toward interactive parallel programs: The on-line steering of a molecular dynamics application, August 1994.

[10] Ada Gavriloska, Karsten Schwan, and Van Oleson. Adaptable mirroring in cluster servers, August 2001.

[11] Steven D. Gribble. Robustness in complex systems, May 2001.

[12] IFIP WG10.4 and IEEE-CS. Dependability.org. www.dependabiliy.org.

[13] Daniela Ivan-Rosu, Karsten Schwan, and Sudhakar Yalamanchili Rakesh Jha. On adaptive resource allocation for complex, real-time applications, December 1997.

[14] Jasmina Jancic, Christian Poellabauer, Karsten Schwan, Matthew Wolf, and Neil Bright. dproc - extensible run-time resources monitoring for cluster applications, April 2002.

[15] Philip Koopman and Henrique Madeira. Dependability benchmarking & prediction: a grand challenge technology problem, November 1999.

[16] J.C. Laprie, editor. *Dependability: Basic concepts and terminology in English, French, German, Italian and Japanese*. Springer-Verlag, 1992.

[17] Jeffrey C. Mogul. Operating systems support for busy Internet services, May 1995.

[18] Brian D. Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin R. Walker. Agile application-aware adaptation for mobility, October 1997.

[19] Christian Poellabauer, Karsten Schwan, Greg Eisenhauer, and Jiantao Kong. Kecho - event communication for distributed kernel services, April 2002.

[20] Rodrigo Rodrigues, Miguel Castro, and Barbara Liskov. Base: using abstraction to improve fault tolerance, December 2001.

[21] Antony Rowstron, Anne-Marie Kermarrec, Peter Druschel, and Miguel Castro. Scribe: The design of a large-scale event notification infrastructure. In *Proc. of the 3rd International Workshop on Networked Group Communication (NGC 2001)*, UCL, London.

[22] D. Scott. Web site availability and scalability: expert user advice. *Gartner Group Research Notes*, December 2000.

[23] Margo I. Seltzer and Christopher Small. Self-monitoring and self-adapting systems, March 1999.

[24] Ariel Tamches and Barton P. Miller. Fine-grained dynamic instrumentation of commodity operating system kernels, February 1999.

[25] Tim Wilson. The cost of downtime. *Internet Week*, July 30 1999.

[26] Rich Wolski, Neil Spring, and Chris Peterson. Implementing a performance forecasting system for metacomputing: The Network Weather Service, November 1997.