SAND2006-5093P

$R^2A^2S^2$: Rapid Routing of Arbitrary, Asynchronous Sequences of Situations

Robert A. Ballance
Sandia National Laboratories, 04328
505.284.9361 raballa@sandia.gov

The world presents many more events than you can process. How do you handle the overload?

Prediction, event detection, attention-focussing, analysis, correlation, response, and recovery are key activities in operating large scale high-performance computing systems. These activities depend on data gathering, analysis, and interpretation. The next generation of high-end systems must necessarily be constructed with usability, maintainability, reliability, availability, and serviceability as primary design goals. The conventional model of ``everything must be working" has become insufficient. We have to design systems so that the applications that run on them can run reliably, in the presence of interruptions, faults, or failures in the underlying hardware and software, including failures in the RAS system itself.

A key role for RAS is to direct the attention of the administrative staff toward situations that are arising, and for which there is not yet any automated response. The RAS systems themselves will have to adapt to changing operational environments and requirements, while managing and analyzing disparate groups of components. The ability to dynamically construct new combinations of analyses in order to detect important correlations among the events is essential. Similarly, we will need organized ways to introduce new automated responses to conditions.

Given the complexities involved, it is insufficient for the RAS subsystems to be static actuators or passive accumulators of data. Instead, an *extensible* architecture is proposed that relies on fast routing among event generators and event interpreters. From such a core architecture, the facilities of an extensible and adaptive RAS system can be constructed.

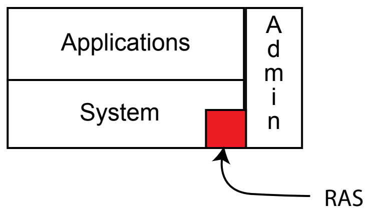

FIGURE 1

Figure 1 illustrates a conventional model of systems: a dependency stack in which "the applications" run on top of "the system." Administrative interfaces sit off to the side, with some assistance from RAS. Interactions between the running application and the system are primarily unidirectional, from the application into the system, such as in system calls. Depending on your point of view, the ADMIN portion is or is not part of RAS.

However, the information flow must be multidirectional: the next generation of systems must support *self-managing* or *autonomic* applications or components by providing situational information back to the application or component. This approach relates back to autonomic computing initiatives such as IBM's Eliza or Sun's N1; what is needed for high-end computing is not just autonomic hardware, but a RAS framework that allows applications to use exploit those capabilities and add new ones of their own.

In many cases it is an arbitrary application, one not envisioned by the original system designers and not the system substrate, that can best respond to a condition. An application, in this sense, is any process running in the system — whether a parallel application code, or an applet injected into the RAS system to explore a specific condition. The RAS infrastructure must be able to inform any application of its present and *predicted future* states so that the application can respond intelligently to changes. These interfaces and protocols are shown as AUTONOMIC INTERFACES in Figure 2. Allowing arbitrary applications to respond to changes in the system employs the end-to-end argument in system design: keep the center simple, and push complexity to the edges.

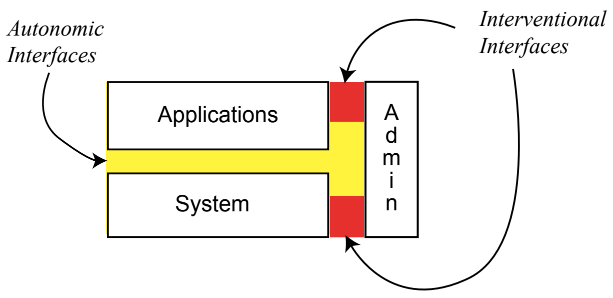

FIGURE 2

The system management portion of the RAS requires control interfaces as well. Applications will be clients of the underlying RAS system. The interfaces require to support effective management of the entire system and its applications appear as INTERVENTIONAL INTERFACES in Figure 2.

A $R^2A^2S^2$ Substrate

One approach is to develop a RAS substrate based on multi-level event routing and processing. The well-known syslog-ng implementation provides a very limited example of this model; it can receive log events, trigger log events, forward log events to other syslog servers, or to programs, or to data stores. Devices and applications are easily modeled as both generators and responders.

1. Events happen. The representation of an event will be the basic unit of communication. Routing events to those components that are interested in them, when they are interested in them, is the crux of operation.

2. Any sequence of one or more events defines a *situation*. Situations can be described using formalisms that cover regular expressions, permutations, projections, and timing constraints. A single event could be a situation, but situations are more complex: such as "Login by USER event, followed within 1 seconds by a logout of the same USER event", or "5 memory failure events from the same DIMM within 1 second", or "Second failure of the same module within 12 days."

3. Events flow to *event routers*. The purpose of the router is to help discriminate *situations* from events, and to notify the relevant responders. Any given event can be routed to multiple responders, since any given event can contribute to several situations. Events can also be stored, and event histories can then be replayed for later analysis.

4. Responders (a.k.a actors, applets, applications, handlers, components, agents, …) respond to situations. Responders register with routers to be notified of situations. In responding, they can forward events, create new events, and even register or deregister other responders. Responders that store events or that allow users to visualize activities are natural additions to the system.

5. A responder can, of course, create new events, which are routed to routers; and routers themselves are special types of responders, so that networks of routers can be constructed. Such networks of routers may be used for aggregation, for failover, for load-balancing, or for load distribution. For example, in the Cray Red Storm RAS system, the blade and cabinet RAS controllers would be (small, low overhead) routers having static routes to other routers.

6. Event generators, routers, and responders are distributed, and will (at times) require authentication mechanisms. Due to the volume of information, support for priorities and quality of service may need to be provided.

The basic job of a router is to route events; it is yet to be determined how much support routers can provide for automatically detecting situations from the event stream. One implementation technique, however, is to allow a responder to register for a situation and let the router create a responder that can detect the situation. Most of the actual discrimination will be performed by responders that parse out sequences of events over time. For example, the situation "Second failure of the same module within 12 days" can be implemented by a special responder that is registered by a 'module-failed-event' responder; this new responder will be activated by any subsequent failure of the specific module being watched.

The lifetime of the responder is arbitrary. Some may be active only long enough to time out, or to receive a single event. Others might need to be actively registered for the lifetime of the system. (Consider the log-all-error-events-to-a-database responder.) Responders can also be created and registered in response to a situation. For example, consider a responder that is created and registered when a login session begins, and completes once the login session ends.

Within the configuration of the system, there will be a minimal set of events and responders that must be present. This configuration can be described, and can be validated by inspecting the state of the RAS. Should the basic configuration checks fail, the RAS can rebuild itself to the minimal state. Similarly, since every interesting event can be tracked, one can implement some portion of the RAS system RAS in the same infrastructure.

From this architecture, one can implement all of the aspects of a RAS system — system operation and administration, logging, analysis, response, focus, and inter-application communication about the current "situation." Logging fits naturally into the architecture as well as hardware transitions and software activities.

There are difficult, but not insurmountable, problems in achieving this vision. The system must be constructed from the start to be scalable, flexible, extensible, maintainable, and understandable. A production deployment will have a lifetime of years. Different organizations will provide components of all types. Automated responders will have to designed to adapt to situations and to provide effective alerts. Mechanisms for prioritizing both event delivery and responses will have to be implemented. Mechanisms for introspection will have to be worked out. Basic API's need to be delivered. Legacy implementations need to be incorporated. One question concerns the role of situations in the system: are they actually represented (realized) as objects that are carried along with events, forming a kind of context that can be inspected? Introspection would be a valuable feature for a RAS.

What th architecture provides is a uniform, and adaptive substrate for all of the interesting facets of a system. It can provide control operations, automatic responses, and attention-direction for the admins, while supporting the application of diverse techniques and algorithms currently under consideration.