Topology Based Automation of Distributed Applications Management

Umesh Bellur
Indian Institute of Tech. Bombay
Powai, Mumbai 400076
India
umesh@it.iitb.ac.in

ABSTRACT

With the widespread use of distributed computing in the enterprise, there have been significant advances in development paradigms for these applications. Server side component models have considerably simplified development and the complexity has now shifted to the operational side of these applications. The increase in operational complexity has reached a point where it is no longer feasible for humans to manage the applications required to run an enterprise. The initial steps to provide self managing applications are now being taken – a paradigm known as "autonomic computing" is in it's infancy of evolution. There have been numerous proposed models of how one achieves self management. In this position paper, we formulate the research problems and basis for "lights out" management of enterprise application environments.

Categories and Subject Descriptors

D.3.3 **[Run time Environments]**: Distributed systems, Autonomic computing, and Analytical methods.

General Terms

Management, Measurement, Performance, Physical Design, Reliability.

1. INTRODUCTION

As more distributed applications become mainstream enterprise solutions, there have been considerable advances in making the development of these applications simpler. The development of server side component models followed by standardization of server side "software containers" to host these components have helped considerably shorten the development lifecycles of large applications. Indeed it is not uncommon to see release cycles of 6 months or less in the enterprise for major features and 3 months or less for minor feature adds.

The impact of these rapid application development paradigms has shifted the complexity from what used to be development to deployment and beyond – tasks that are commonly handled by the IT Operations staff in the enterprise. Once the application has been developed, the first task would be to map it to a physical architecture given the expected workloads and the availability of shared physical resources (CPU, disk, network bandwidth etc.). Once resource mapping is done, the various resources need to be configured with the appropriate parameters to handle the application. This in itself is a task of great complexity not only because of the dependencies between the various components making up an application but also because one needs to map any QoS requirements of the application (such as response times and uptime) to the selection of the different physical components that the application will run on. For example, network QoS may have to be negotiated appropriately since network communication quality can have a significant impact on application performance of distributed applications. The complexity also arises from the numbers of parameters that have to be tuned on resources such as application servers and relational databases. The modern J2EE[1] application server has over 300 parameters that have to be tuned in order to extract the best value.

Subsequently, monitoring the application with a view to resolving faults that may occur as well as keeping the performance tuned in spite of varying workloads is also a daunting task – one that is amplified by the presence of several such applications running on the enterprise's wide and/or local area network. Empirical evidence suggests that it is impossible to manually handle and automating these tasks is a necessity.

Of late, there has been an increased focus on "autonomic computing" techniques – techniques that determine how application environments can configure and heal themselves in the event of problems. For example, an application server (or middleware server) can have over a hundred different parameters that have to be tuned.

In this paper, we first present an application management architecture that spans resource discovery to fault detection, isolation and correction. We are in the process of realization of this architecture and this paper is work in progress towards the goal of what is termed zero-touch or lights out management[2]. This is part of the LAMDA (**L**ights-out, **A**utomated **M**anagement of **D**istributed **A**pplications) project being done at IIT in conjunction with the industry.

WOSP '04, January 14-16, 2004, Redwood City, CA.

[1] J2EE is a trademark of Sun Microsystems and denotes the server side Java component architecture commonly used to build enterprise applications today.

[2] Lights Out management is a term commonly used in the IT industry to indicate that no human is needed to manage these applications.

2. The LAMDA Vision

There are several facets to autonomic computing all of which form part of the LAMDA vision.

a. Physical Design and deployment – Self Configuration. There are two aspects to this – static and dynamic. Static design lays out certain constraints on location of the application components and maps it initially to a physical topology. The dynamic version ensures that these constraints continue to be met and may move application components, add or remove computing resources and reconfigure the infrastructure.

b. Root Cause Isolation and correction - Self Healing. Self healing can be for the purposes of correcting a structural constraint or property that has been broken such as those related to performance, availability or capacity.

c. Self Protection – Related to the second facet, this is for the purposes of healing a security breach that has occurred. The techniques and the basis for self protection are often very different from those used for self healing and so will be considered separately.

As a part of this effort (especially part a), we have also developed meta models for describing application QoS parameters and resource needs which we use in trying to come up with the physical design.

2.1 The Basis of LAMDA

2.1.1 Structural Basis - Topology

The starting point for self-healing or self configuration is to know one self and so determining the topology of the application in relation to its execution environment is critical. An application cannot be deployed without knowledge of the various components that make it up. Both the static parts of the component (viz, it's packaging) as well as it's physical footprint need to be well understood for problem isolation and correction.

Topology therefore is a description of:

a. The infrastructure (both physical such as compute servers as well as logical such as server component containers), its configuration and its dependence on the underlying network.

b. The static view application components and their configurations.

c. The dynamic or run time view of application components that execute on the infrastructure. This specifies the physical footprint that the component exhibits at run time. For example, an EJB can be deployed on several J2EE containers either as a cluster or singly.

d. Dependencies that exist between application components, between application components and infrastructure (software, hardware and network).

Topology is a realization of the meta-model that characterizes applications and their execution environments and provides a canonical language for common understanding of what an application is and what it depends on. Every tool in the LAMDA arsenal works off of topology. Since the topology of a distributed shared execution environment is constantly changing (applications are being added, removed or updated, machines are upgraded or added, the network is being tuned etc.), we need a process that will keep up-to-date the topology of the existing environment including any applications that are currently executing on it.

2.1.2 Analytical Basis

In order to have a predictive model of both capacity management as well as potential failures, we need an analytical model of an application and its execution infrastructure that we can solve under the constraints specified by the needed application QoS.

For the purposes of self configuration as it relates to performance tuning and capacity management we are using Hierarchical Queuing Petri Nets (HQPN) to model our environment. HPQNs are a variation of Colored General Stochastic Petri Nets and stochastic queuing models where we can build hierarchies of such Petri nets recursively. Every place can be attached to a Queue to represent scheduling policies and waits. The hierarchy is built up by folding the sub Petri net to represent a single place which has a timed wait. HQPNs have been employed in similar situations to analyze application performance and the component model of deployment is particularly well suited to be modeled using HPQNs. For further information on HPQNs, we refer the reader to [15]. They translate to their underlying Markov chains which can be solved using well understood methods.

The analytical basis for self healing however is still in the formative stage but we are leaning towards using multi-agent architectures (MAS) coupled with distributed correlation algorithms that correlate across the network, compute and software infrastructure layers. MAS gives us the ability to decentralize decision making as it related to root cause isolation and also adds the notion of machine learning which is needed in trying to isolate root causes from a variety of patterns that occur in these complex environments.

3. Current Status

This project was born out of the experience of several system administrators who had the first hand experience of setting and managing service QoS on multiple applications in a shared data center environment. Since then we have added an analytical flavor to the application management process architecture.

We have currently implemented a functional Discovery subsystem which works off the meta-model described in earlier sections. This tool does auto discovery of a networked environment and can discover and map the topology of:

a. Layer 2 and 3 (IP) Networks including VLANs, VPNs, Firewalls and Load Balancers

b. Compute layers consisting of heterogeneous operating systems (SUN Solaris, Linux etc.) and classes of machines.

c. Software infrastructure such as Apache Web servers (Version 1.3+), J2EE Application Servers (JBOSS Version 3.0+) and Oracle Databases (Version 8 and 9).

d. Application components such as Servlets/JSPs, Enterprise Java Beans and DB Schemas along with their interdependencies.

The starting point for this tool is a range of IP Addresses which serves as the bounds of discovery. We have also performance benchmarked this tool up to a 300 server data center environment and performance is more than adequate at about 15 seconds for a 100 servers with linear increase. We have also proved that Discovery consumes less than 3% of the system resources to run.

We are in the process of dealing with the other problems described earlier and are putting together analytical models for this environment.

4. SUMMARY

To tackle the growing complexity of managing distributed/networked applications, we have proposed management architecture for autonomic computing of such environments. The LAMDA architecture revolves around the environment's topology for which we have developed a meta-model.

Although the work is ongoing, this paper states our position on the architectural approaches that are required to deal with the issues holistically. We feel that there will be significant benefit to interact with the other researchers in the area who may be taking other approaches and that the exchange of ideas will benefit all concerned.

5. REFERENCES

[1] Bigus, J. P., et al. "A Toolkit for Building Multiagent AutonomicSystems", http://www.research.ibm.com/journal/sj/413/bigus.html, 2002.

[2] Blair, G., et al., "Reflection, Self-Awareness and Self-Healing in OpenORB", ACM WOSS", 9-14, Nov., 2002.

[3] Dabrowshi, C. and Mills K., "Understanding Self-healing in Service-Discovery Systems", ACM WOSS, Charleston, SC, USA., 15-20, Nov., 2002.

[4] Dashofy E. M.m et al., "Towards Architecture-based Self-Healing Systems", ACM WOSS, Charleston, SC, USA., 21-26, Nov., 2002.

[5] Fox A. and Patterson, D., "When Does Fast Recovery Trump High Reliability?", *Proceedings of the EASY 2002*, San Jose, CA, October 2002.

[6] Ganek, A., "A letter from Vice President, Autononomic Computing, Alan Ganek" http://www-3.ibm.com/autonomic/letter.shtml, 2002.

[7] Garlan, D. and Schmerl, B., "Model-based Adaptation for Self-Healing Systems", ACM WOSS, Charleston, SC, USA., 27-32, Nov., 2002.

[8] George S., et al., "A Biologically Inspired Programming Model for Self-Healing Systems", ACM WOSS, Charleston, SC, USA., 102-104, Nov., 2002.

[9] IBM paper-1: "IBM autonomic computing challenges note: academic focus article: challenges", http://www.research.ibm.com/autonomic/academic/challenges.html, 2002

[10] Mikic-Rakic, M., et al., "Architectural Style Requirements for Self-Healing Systems, ACM WOSS, Charleston, SC, USA., 49-54, Nov., 2002.

[11] Patterson, D., et al., "Recovery Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies", In *Proceedings of the UC Berkeley Computer Science Technical Report UCB/CSD-02-1175*, Berkeley, CA, March 2002.

[12] Tivoli software, "Autonomic Computing: The Value of Self Managing Systems", http://www.tivoli.com/news/features/oct2002/autonomic.html, 2002.

[13] Vaidyanathan, K., Selvamuthu, D., and Trivedi, K. S., Analysis of Inspection-Based Preventive Maintenance in Operational Software Systems, Intl. Symposium on Reliable Distributed Systems, SRDS 2002, Osaka, Japan, October 2002

[14] Probability and Statistics with Reliability, Queueing and Computer Science Applications, Kishore S. Trivedi, ISBN 0-471-33341-7, John Wiley and Sons.

[15] F. Bause, P. Buchholz and P. Kemper – QPN Tool for the Specification and Analysis of Hierarchically Combined Queueing Petri Nets. Quantitative Evaluation of Computing and Communication Systems, Lecture Notes in Computer Science No. 977, Springer-Verlag, 1995