Quartermaster—A Resource Utility System

S. Singhal, M. Arlitt, D. Beyer, S. Graupner, V. Machiraju, J. Pruyne, J. Rolia, A. Sahai, C. Santos, J. Ward, X. Zhu
HP Laboratories
1501 Page Mill Road, MS 1125
Palo Alto, CA 94304
USA
{sharad.singhal, martin.arlitt, dirk.beyer, sven.graupner, vijay.machiraju, jim.pruyne, jerry.rolia, akhil.sahai, cipriano.santos, jward, xiaoyun.zhu}@hp.com

Abstract

Utility computing is envisioned as the future of enterprise IT environments. Achieving utility computing is a daunting task, because enterprise users have diverse and complex needs. In this paper we describe Quartermaster, an integrated set of tools that addresses some of these needs. Quartermaster supports the entire lifecycle of computing tasks - including design, deployment, operation, and decommissioning of each task. Although individual components of this lifecycle have been addressed in earlier work, Quartermaster integrates them in a unified framework using model-based automation. All tools within Quartermaster are integrated using models based on the Common Information Model (CIM), an industry-standard model from the Distributed Management Task Force (DMTF). The paper discusses the Quartermaster implementation, and describes two case studies using Quartermaster.

Keywords

Utility computing, resource allocation, policy-based resource composition, CIM

1. Introduction

The increasing complexity of IT environments has made them difficult to manage in a cost effective manner. This has led to a push within enterprises for consolidating IT environments into shared pools of resources, which are partitioned dynamically to provide resources to applications. Within this context, we define utility computing as the ability to provide complex computing environments on-demand to IT users. Achieving utility computing is difficult because the needs of enterprise users are complex. Each application running within the enterprise has unique assumptions, each enterprise has different policies that are associated with its applications, and each user brings a different set of requirements to the application. Providing infrastructure that supports these diverse requirements is not a straightforward task. While techniques such as virtualization or load-balancing are necessary for utility computing, they are not sufficient. To be successful, a utility computing system must support design, deployment, and management of arbitrary applications while dealing with their frequently competing requirements for resources; accommodate both user and operator policies on how infrastructure is used; deal with upgrades of both the

To be published in the 9th IEEE/IFIP International Conference on Integrated Management, May 15-19, 2005, Nice, France.

infrastructure and the applications; and maintain a high level of automation to reduce errors and manage costs.

We address utility computing from the perspective of IT administrators. That is, given user-specified requirements for a complex application, how can the corresponding system specification be automatically created and the system provided to the user on-demand? Our goal is to provide the administrators with tools that support the entire lifecycle of a computing task—including the design, deployment, operation, and decommissioning of that task while maintaining flexibility, agility, and cost efficiency within the utility through automation. Our approach is using model-based automation—creating models of the IT environment and the desired application and creating a set of tools that use those models to automate IT tasks. In this paper, we describe Quartermaster, an integrated set of tools and technologies targeted at this problem.

The remainder of the paper is organized as follows: Section 2 describes the functionality provided by Quartermaster, and discusses the various tools that provide those functions. Section 3 presents the Quartermaster software architecture and describes how the tools are integrated using a model-based approach. Section 4 provides two case studies where we are currently using Quartermaster. Section 5 describes related work, and we conclude the paper with a summary of our contributions in Section 6.

2. Quartermaster Tools

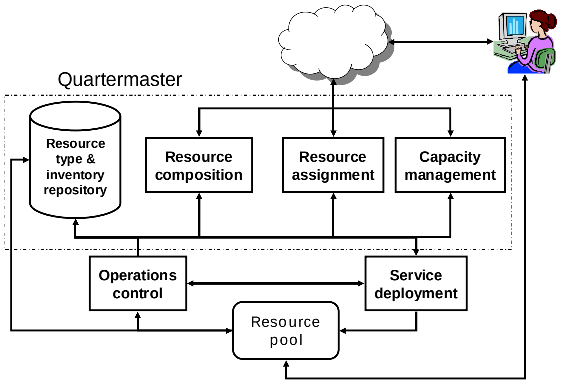

Figure 1: Overall architecture of Quartermaster

Figure 1 shows the overall functional architecture of Quartermaster. A repository maintains models of the different resources known to Quartermaster, as well as an inventory of resource instances available to Quartermaster. It provides Quartermaster tools with a uniform way of interacting with the resources. IT operators and users can interact with Quartermaster tools using either visual or programmatic interfaces. Currently, the tools provide users with the ability to compose IT resources into desired configurations, to manage the capacity of the underlying resource pools, and to schedule and allocate resources to applications. The Quartermaster tools can be integrated with service deployment capabilities offered by other technologies [1] as well as with management tools that provide operations management and control capabilities [2] to provide an end-to-end view of resource management within the data center. In the next few subsections, we describe the capabilities offered by the tools in more detail.

2.1. Policy-driven Resource Composition

A variety of rules need to be followed during system configuration to ensure correct operation. For example, when operating systems are loaded on a host, it is necessary to validate that the processor architecture assumed in the operating system is indeed the architecture on the host. Similarly, when an application tier is composed from a group of servers, it may be necessary to ensure that all network interfaces are configured to be on the same subnet or that the same version of the application is loaded on all machines in the tier. To ensure correct behavior of a reasonably complex application, several hundred such rules may be necessary. This is further complicated by the fact that a large fraction of these rules are not inherent to the resources or the application, but depend on preferences (policies) provided by the system operator. The resource composition tool provides users with the capability to generate system configurations that comply with such rules.

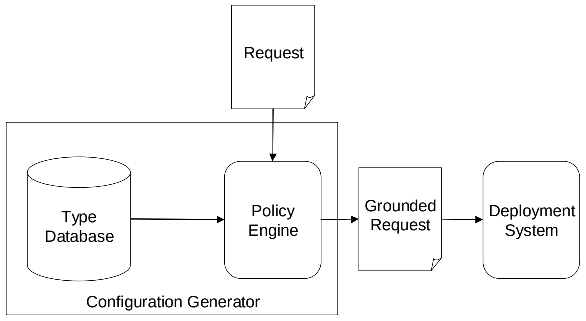

Figure 2: Resource configuration process

The tool is exposed to the user as a “drag-and-drop” graphical user interface (GUI), which can be used to design complex environments "on-the-fly" from components such as software, servers, and storage devices. The tool allows operators to define templates for a broad range of resources and provide them to users as resource templates on the GUI. The user can then select components from the palette and design the desired environment. For example, the user can simply drag in the icon representing an e-commerce site into the design panel, add policy constraints representing specific requirements (e.g., the number of transactions per second the site must be capable of supporting, the size of the database, etc.), and ask the tool to generate a configuration based on the template.

Figure 2 shows the high level structure of the composition tool. The tool treats the user’s request (which may be minimally specific) and the policy rules embedded in the request as a goal to be achieved. It fetches the component model definitions from the model repository and selects both classes and attribute values within those classes such that all configuration rules are satisfied. This generates a “grounded” request specification containing an instance-level description that is handed to the deployment system [1] for instantiation. Additionally, deployment activities are also modeled as classes, and the tool selects the appropriate activities to generate a workflow [5] for deployment.

The policy engine [3] in the composition tool treats system configuration as a constraint satisfaction problem [4]. Complex environments requested by users are treated as higher-level resources that are composed from other resources. Configuration rules are embedded as constraints in the various resource models, specified by the operators of the resource pool, or by users as part of the requests for resources. The constraints consist of predicates represented in first order logic with linear arithmetic. The policy engine and the configuration processes are described in more detail in [3], [5].

2.2. Proactive Capacity Management

Once a configuration has been generated, the user submits the design to a capacity manager [6]. Managing capacity of large resource pools in enterprise environments is a challenging problem because, unlike batch environments, applications in enterprise environments are long running but have time-varying resource demands. The capacity manager ensures that sufficient capacity is available to support the longer term aggregate demands of applications.

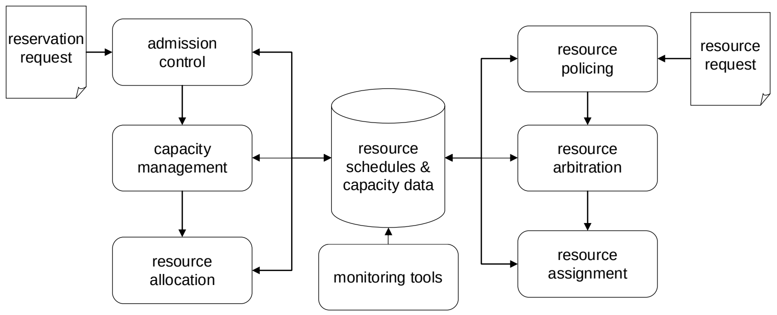

Figure 3: Capacity management process

The capacity manager includes scheduling and capacity management algorithms [7], [8], [9] that can take such fluctuations into account. The algorithms rely on historical traces of application demand to forecast future resource requirements. If no historical information is available, the application can be initially provisioned for its anticipated peak demand, and its behavior can be characterized over time. As shown in Figure 3, the capacity manager includes several components: admission control, capacity management, resource allocation, policing, arbitration, and assignment. These components address the following questions: which applications should be admitted (i.e., permitted to make resource reservations), how much resource capacity must be set aside to meet their needs, which applications are currently entitled to resource capacity on-demand, which requests for resource capacity will be satisfied, and which units of resource capacity should be assigned to each application.

Admission control and resource acquisition processes are illustrated on the left in Figure 3. Admission control decides whether an application will be accepted. It relies on the resource allocation system to determine which resource pools have sufficient capacity to satisfy the demands of the application. Once specific resource pools are chosen, reservations are made and reflected in the capacity management plan. Note that individual resources within the pool are not assigned to the application at this time (see Section 2.3). Requests for resource capacity from admitted applications are illustrated on the right in Figure 3. Resource capacity requests are batched by the capacity manager so that tradeoffs can be made regarding which requests for resource capacity are satisfied. Policing mechanisms verify that an application's request for capacity is within the bounds of its SLA. If it is, then the request is entitled to the capacity. If demand exceeds supply, then arbitration mechanisms are used to decide which requests are satisfied. Requests that are to be satisfied are assigned resources. This is discussed in more detail in the next Section.

2.3. Optimized Resource Assignment

If capacity is available, the capacity manager works with a resource assignment system [10], [11] to assign the actual instances of resources for the application. Manual (or simple heuristic) approaches to resource assignment work when all resources are equivalent (e.g., in a cluster), or when the resource pool is small. With complex topologies, it is easy to create bottlenecks in the shared resources when using such approaches, resulting in failure to meet application requirements even when capacity is available in the data center. The assignment system automates selection of the servers within the data center fabric for deployment of the application.

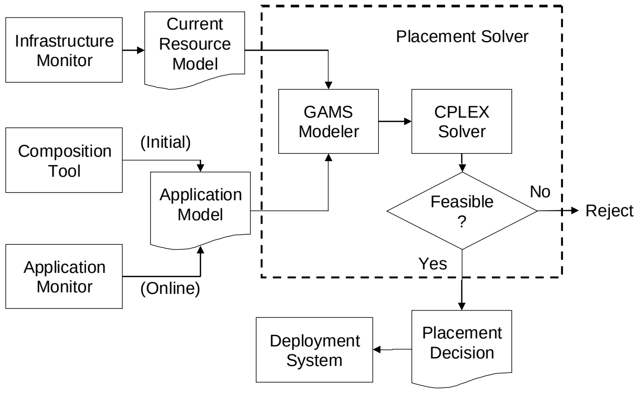

Figure 4: Resource assignment process

Figure 4 shows the process used by the assignment system. The solver requires two models as input: a resource model and an application model. The resource model describes the fabric topology and the resource capacities. The application model defines the application topology and its resource requirements. The infrastructure monitor tracks the resource inventory, including the connection topology and available capacity. The monitor maintains an up-to-date model of the current state of the environment using information from the resource inventory and monitoring tools (such as OpenView). The composition tool (Section 2.1) maps the application's high-level QoS goals into an initial application model that represents the low-level processing, communication and storage requirements on the physical resources. The constraints and the objective function required by the solver are dynamically generated from these parameters using the modeling language GAMS [12], and fed into the CPLEX solver [13]. The latter checks the feasibility of the problem, and finds the optimal solution among all feasible solutions. The detailed models required by the assignment solver are described in [11].

3. Quartermaster Software Architecture

All tools within Quartermaster are integrated through an information model that describes management concepts such as resources, relationships between resources, and policies that apply to them. Figure 5 shows the software architecture for Quartermaster. Core to the architecture is a model repository that stores the information model as well as an inventory of managed elements that conform to that model. Surrounding the repository are tools that are necessary to manage the model and the inventory. Model instances and metrics are populated using a provider framework that integrates both monitoring data and instance discovery from management frameworks such as OpenView. Quartermaster tools are layered on top of the model repository.

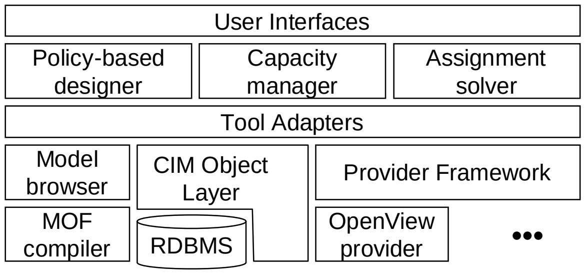

Figure 5: Quartermaster software architecture

3.1. Information Model

Central to Quartermaster is a common information model that is shared among the tools. The information model captures the concepts and terminology required for managing systems in a standard manner, thereby allowing individual tools to interoperate through the model. In other words, the model serves as the "integration bus" as one tool updates the model and another tool reads from it.

We have chosen to use the Common Information Model (CIM) [14] defined by Distributed Management Task Force (DMTF) [15] as the basis for the information model in Quartermaster. CIM is an object-oriented model that defines how managed elements - ranging from physical devices and computer systems to applications - must be modeled to facilitate integration between management systems. Not all management concepts that were required by the Quartermaster tools were modeled in CIM. In particular, we had to augment the CIM model with the following concepts – *policies* or constraint rules as required by the policy-based composition tool, *contexts* to represent potential, future, and alternative choices for configuring systems, and *metric providers* to represent information about sources of measurement data available from managed systems. These three additions were made by defining new classes conformant to the CIM meta-model as described below. In the discussion, classes defined as part of the standard CIM model are shown with a prefix of "`CIM_`", while those added in Quartermaster are prefixed with "`QM_`".

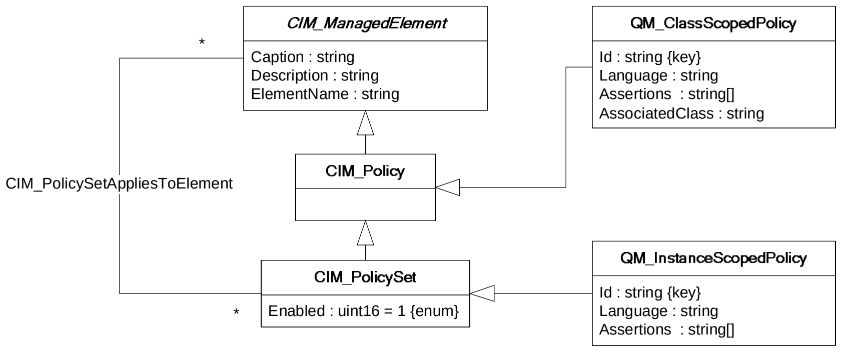

Figure 6: Policy extensions to CIM model

A *policy* in Quartermaster, as explained in Section 2.1, is a constraint attached to a class defined in the information model or to an instance of a class. A model describing these two concepts is shown in Figure 6. Policies attached to a class have to be obeyed by all instances of that class (as well as any refined classes derived from that class). We call them *class scoped policies*. Policies attached to an instance have to be obeyed by that instance alone and are called *instance scoped policies*.

The `AssociatedClass` property in `QM_ClassScopedPolicy` defines the class in the information model that is referred to by the policy. The association between `CIM_PolicySet` and `CIM_ManagedElement` defines the instances to which an instance scoped policy is attached.

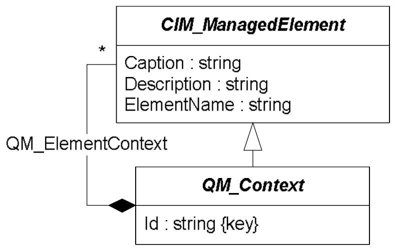

Figure 7: Context extension to CIM model

CIM classes usually model managed elements that are present in the environment. This is not always the case, however, with Quartermaster tools. For instance, the policy-based composition tool may be used to design a database that may or may not be implemented. Similarly, the capacity management tool may be used to perform a what-if analysis on a set of servers that are non-existent. Because tools communicate using the models, such scenarios require that managed elements are instantiated in the repository only as part of a fictitious context, but not as part of the real environment. In other words, we need the capability to represent alternative configurations - that may or may not have realizations - at the same time in the repository. To solve this problem, we have defined a new class called `QM_Context` that groups all the managed elements relevant to a particular context. All tools in Quartermaster operate on a given context and manipulate managed elements only within that context. Figure 7 shows the model for a context.

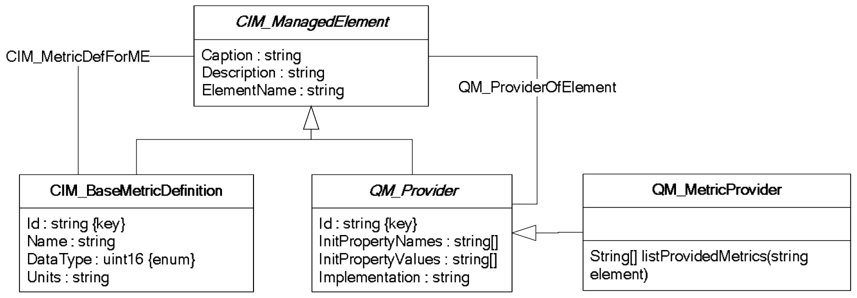

Figure 8: Metric provider extension to CIM model

Our third extension to CIM models was to model a metric provider. CIM defines what metrics are available on a given managed element. However, it does not define how to obtain those metrics. Since some of the tools in Quartermaster rely on traces of metrics collected by instrumentation sources (e.g., CPU and memory utilization data collected every 5 minutes on every server), we had to augment the CIM model with metric providers. This allowed Quartermaster tools to pull the required metric traces automatically whenever needed. Figure 8 shows the metric provider model.

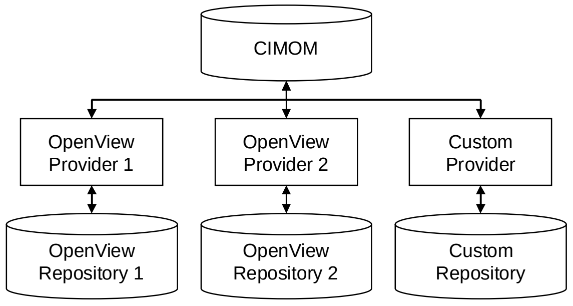

Figure 9: Integrating existing sources of instance data into CIMOM

All the classes defined in CIM, along with the above three extensions, are loaded into a repository - usually referred to as CIM Object Manager or CIMOM. We have implemented a CIMOM using a relational database (MySQL). The API on CIMOM can then be used to manipulate the model or its instances. Information about instances is made available to the CIMOM through a series of instance providers that adapt data from discovery tools such as OpenView as shown in Figure 9.

3.2. Integrating Quartermaster Tools

Each tool in Quartermaster operates on a portion of the information model. This raises two issues: First, how does Quartermaster integrate a tool that does not understand CIM (and our extensions), and second, how do multiple tools integrate with each other?

Quartermaster communicates with tools in a loosely-coupled manner. The tools expect information (and provide output) in tool-specific formats. Each tool is integrated using a tool manager, which uses an output adapter to convert model information from the Quartermaster repository to the format required by the tool, and converts the tool output using an input adapter to a form appropriate for the model repository. An example of this software pattern is shown in Figure 10, where the interaction with the resource assignment system is shown. The placement tool manager (the Placement Designer) is asked to place an entity (the grounded request). It uses the output adapter to retrieve information required for the placement (see 2.3, [11]) from the repository and converts it into an XML format required by the placement tool. It then uses an HTTP POST operation to send the information through a web server to the CPLEX solver. The assignment information is received by the Placement Designer in an XML format, and used by the input adapter to make the appropriate associations in the repository.

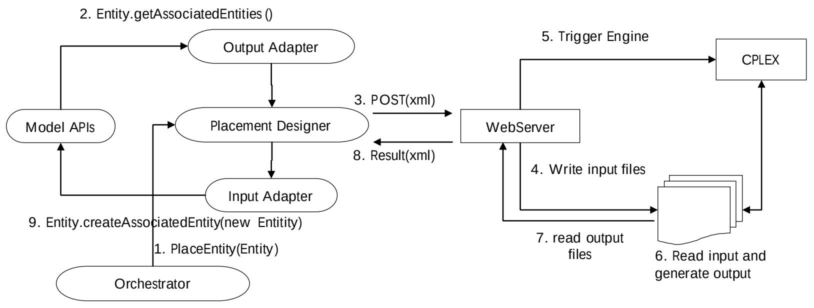

Figure 10: Interaction with resource assignment tool

To explain how multiple tools integrate with each other, let us take an example. Say, we have to design an e-commerce site and allocate resources to run that site. This is accomplished in Quartermaster through the use of two tools – policy-based system composition tool to design the e-commerce site and the resource assignment tool to allocate compute resources to each of the components of that design.

To use the policy-based system composition tool, models of the e-commerce site have to be created by operators along with the policies that govern how such a site may be constructed to meet the given objectives. The user then inputs instance-specific constraints (e.g., number of transactions/sec required) as part of the request. This request is embedded in a context that contains an instance of the e-commerce site class (without the associated class hierarchy, or attribute values filled in). All the desired objectives are expressed as instance-scoped policies on this instance. The composition tool uses this context and outputs a transformed context containing a fully expanded instance of the e-commerce site with the values for attributes and composition structure filled in. The input to the assignment tool is a context that contains two instances – one representing the topology of the infrastructure and the other representing the topology of the application (the expanded e-commerce site instance from the composition tool). The assignment tool transforms this context by adding the “hosted on” relationships between application topology nodes and infrastructure topology nodes. To summarize, each tool operates on and transforms a context. Multiple tools integrate with each other by taking the results from one context and using them as inputs in a different context.

There are several benefits to our approach of tool integration. The whole design and assignment of resources in the above example is grouped into individual contexts. This means that an operator can inspect the design resulting from the composition tool, try different designs, and compare the results. Similarly, the operator can inspect the assignment made by the assignment tool for each alternate design, and keep the ones he/she liked. Throwing away the results of a tool is as simple as deleting a context (which also deletes all class instances contained in that context). This provides flexibility in chaining tools as well as in trying multiple alternatives, which is particularly important in decision-making tools such as those in Quartermaster.

4. Experience using Quartermaster

We have integrated the components within Quartermaster into a test-bed that we are using to further explore the component algorithms in our research. We have used this test-bed in a number of initiatives to understand its capabilities within service deployment, configuration, and lifecycle management scenarios. We briefly mention two of these initiatives below, as examples of problems that may be tackled by Quartermaster technologies.

HP's Shared Application Server Utility (SASU): Like many large companies, HP maintains and operates hundreds of internal business applications. Each of these applications has traditionally required its own server. HP IT is currently consolidating its J2EE servers into a shared utility that will support the needs of J2EE applications, and provide the application server platform as a utility to groups within HP. This would reduce the licensing, support, management, and hardware costs associated with these applications.

The service is hosted on clusters of HP-UX servers and exploits HP-UX workload management features [25] to co-host multiple applications on the same server while providing them with resource guarantees. The capacity management components of Quartermaster are being used as part of this process. The Quartermaster capacity manager supports admission control exercises, recommends which server(s) are best suited for supporting a new application, indicates which services should share servers, and guides the setting of configuration parameters needed for the workload manager that controls the fine grain assignment of server CPU shares to the services.

Figure 11 shows a typical analysis output from the capacity management tool. The output shown is based on three months of data from two production servers, each with 16 CPUs. The servers host 37 applications with 17 applications assigned to server 1 and 20 to server 2. As shown in Figure 11, if the servers were sized based on the peak demands of each application, a total of 42.7 and 36.7 CPUs would be necessary to support the applications, respectively. The figure shows that by sharing, the servers require only 11.0 and 13.7 CPUs respectively.

Next, we consider adding a new application to these servers. The application is projected to have a peak demand of 2.9 CPUs and a mean demand of 0.3 CPUs.

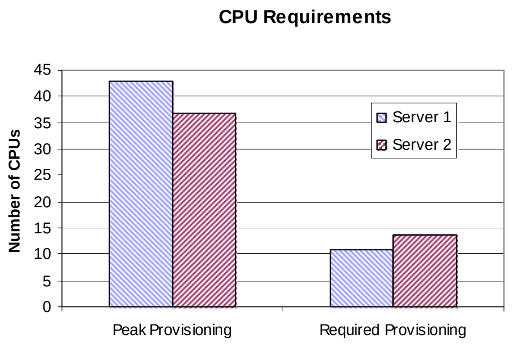

Figure 11: Typical analysis output of capacity management tool for SASU

Figure 12 shows the result of the planning exercise. Assigning the application to server 1 increases its required number of CPUs from 11.0 to 11.4. Assigning it to server 2 increases its required number of CPUs from 13.7 to 13.8 CPUs. Though the new application achieves better sharing on server 2, the capacity manager assigns the application workload to server 1 to better balance the workloads.

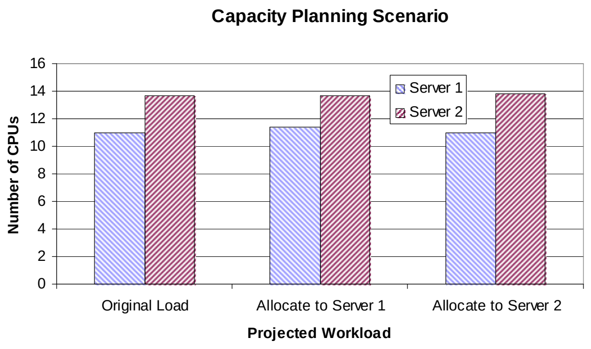

Figure 12: Typical capacity planning scenario within SASU

Financial Services IT Consolidation Initiative: We are currently working with one of HP’s large financial services customers on an IT consolidation initiative. As part of this initiative, we are exploring how PC users could be moved to terminal-based “virtualized” desktops hosted within centralized compute clusters and storage facilities. For testing purposes, we have replicated this environment within our lab using bladed servers. We are using Quartermaster tools within this test environment to create an integrated model-based view of the entire system, including the resources, services, and clients using the environment. Using these models, we plan to explore how resources could be automatically configured and scheduled for clients.

Figure 13 shows response times measured on our test-bed for typical office tasks for the virtualized desktop when different numbers of virtual machines are hosted on a server. Response times from these and other applications are captured in the resource templates used to generate configurations for the virtual desktops, and to decide the number of virtual machines that can be hosted on a server. This in turn frees operators to simply specify user requirements in terms of the mix of applications required by a user and the acceptable application-level performance needed.

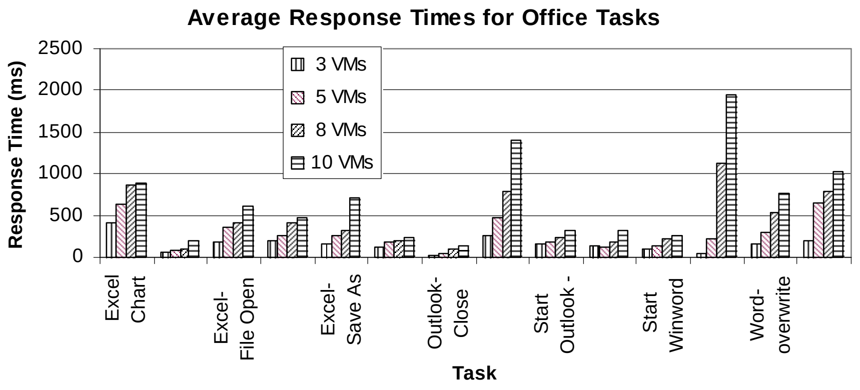

Figure 13: Typical times observed for various office tasks in a virtualized desktop using different number of virtual machines

In both case studies, feedback from the system operators has been positive. In particular, operators have found that Quartermaster tools provide an integrated view of the system that is otherwise not available, and focus attention on data that is at the right level of abstraction. For example, within the virtualized desktop, operators can define policies associated with different classes of users, and schedule resources based on the user class. Additionally, operators have found the framework useful because it enables them to quantify improvements (e.g., utilized capacity or time-to-deployment of applications) that are otherwise difficult to evaluate. Within the SASU environment, operators have found approximately 25% time savings for capacity management exercises.

5. Related Work

Many of the individual problems tackled within Quartermaster have been described in the literature. System composition has been explored within the artificial intelligence [16] and software engineering [17] communities. A rich literature exists on problems of scheduling resources to applications in the computer science community, as well as in the high performance computing community [21]. Similarly, the statistics [22] and operations research communities [23] have studied algorithms for modeling time varying quantities and optimization algorithms respectively. Finally, system modeling has been subject to much research within the software engineering field [24]. Quartermaster brings together knowledge from these diverse fields into tools that can be applied to solve IT automation problems.

The use of CIM in policy validation is described in [18]. A facility for handling rules within a CIMOM is described in [20]. Automation of change management processes using optimization is described in [19]. Quartermaster differs from such previous work in its effort to bring together diverse tools in a model-based approach to integration and its focus on design-time tasks required by operators, as opposed to run-time management.

6. Summary

Quartermaster seeks to create an IT resource utility, where complex applications can be provisioned for IT users on-demand. Quartermaster contains an integrated set of tools that provide users with the ability to compose complex environments, manage capacity within resource pools, and allocate resources from those resource pools to applications and users. The tools are integrated using a model-based approach that relies on the CIM standard for modeling the environment. Individual tools communicate by changing the elements within a shared model.

We are using these tools in a number of initiatives both within HP and with HP customers to test their applicability within different use cases, to obtain feedback and experience from their use, and to refine the tools using this experience. In addition, based on user feedback from our case studies, we plan to include other capabilities that would provide Quartermaster with:

- the ability to measure the behavior of the designed system at run-time and automatically adapt the design to maintain it within user-specified bounds;
- measurements of application behavior to enable iterative improvement and refinement of the component types; and
- real-time data from the resource pool to our resource allocation and placement algorithms, so that inaccuracies in the parameters specified in the design do not accumulate in practice.

The Quartermaster architecture provides us a framework within which these (and other) capabilities are integrated.

References

[1] SmartFrog http://www.smartfrog.org/

[2] OpenView http://openview.hp.com/

[3] A. Sahai, S. Singhal, V. Machiraju, R. Joshi, “Automated Policy-Based Resource Construction in Utility Computing Environments,” 2004 IEEE/IFIP Network Operations and Management Symposium, Seoul, Korea, April 2004.

[4] C. Flanagan, R. Joshi , X. Ou,, J. Saxe, “Theorem Proving Using Lazy Proof Explication,” In Lecture Notes in Computer Science, Springer-Verlag Heidelberg, Volume 2725, pp. 355-367, Jul 2003.

[5] A. Sahai, S. Singhal, R. Joshi, V. Machiraju, “Automated Generation of Resource Configurations through Policies,” IEEE 5th International Workshop on Policies for Distributed Systems and Networks, YorkTown Heights, June 2004.

[6] J. Pruyne and V. Machiraju, “Quartermaster: Grid Services for Data Center Resource Reservation,” Global Grid Forum Workshop on Designing and Building Grid Services, October 8, 2003, Chicago, Illinois, USA

[7] J. Rolia, X. Zhu, M. Arlitt and A. Andrzejak, "Statistical Service Assurance for Applications in Utility Grid Environments," IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems, Ft. Worth TX, October 2002.

[8] J. Rolia, X. Zhu and M. Arlitt, "Resource Access Management for a Resource Utility for Commercial Applications," IEEE/IFIP Int. Symposium on Integrated Network Management, Colorado Springs, CO, March 2003.

[9] J. Rolia, A. Anderzejak, and M. Arlitt, "Automating Enterprise Application Placement in Resource Utilities," IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, LCNS 2867, M. Brunner and A. Keller (eds), pp. 118-129.

[10] X. Zhu and S. Singhal, "Optimal Resource Assignment in Internet Data Centers,"- IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems, Cincinnati, OH, August 2001.

[11] X. Zhu, C. Santos, J. Ward, D. Beyer, S. Singhal, "Resource Assignment for Large Scale Computing Utilities using Mathematical Programming," HPL Tech. Rep. HPL-2003-243, November 2003.

[12] GAMS, www.gams.com

[13] CPLEX, www.ilog.com

[14] CIM Modeling http://www.dmtf.org/standards/standard_cim.php

[15] DMTF: http://www.dmtf.org

[16] F. Brazier, C. Jonker, J. Treur, "Principles of Compositional Multi-Agent System Development," Proc. of the IFIP'98 Conference IT&KNOWS'98, J. Cuena (ed.), Chapman and Hall, 1998

[17] C. Lucas and P. Steyaert. "Research topics in composability," Proc. of the CIOO Workshop at ECOOP, Linz, July 1996

[18] L. Lymberopoulos, E. Lupu, M. Sloman, "PONDER policy validation in a CIM and differentiated services framework," Proc. of the IFIP/IEEE Network Operations and Management Symposium, Seoul, Korea, 2004, pp. 31-44.

[19] A. Keller et. al., "The CHAMPS System: Change Management with Planning and Scheduling," Proc. of the IFIP/IEEE Network Operations and Management Symposium, Seoul, Korea, 2004, pp. 395-408.

[20] S. Nakadai, M. Kudo, K. Konishi, "Rule-based CIM Query Facility for Dependency Resolution," Proc. of the 15th IFIP/IEEE International Workshop on Distributed Systems, Davis, CA, 2004, pp. 245-256.

[21] S. Chang, J. A. Stankovic and K. Ramamritham, "Scheduling algorithms for hard real-time systems: a brief survey," in J. A. Stankovic and K. Ramamritham (eds), Hard Real-Time Systems: Tutorial, IEEE, 1988, pp. 150-173.

[22] S. Levinson, "Statistical modeling and classification," in Survey of the State of the Art in Human Language Technology, Cambridge University Press, 1996.

[23] C. Harvey, "Operations Research: An Introduction to Linear Optimization and Decision Analysis," Elsevier Science, 1979

[24] A. Felfernig, G. E. Friedrich et al. UML as a domain specific knowledge for the construction of knowledge based configuration systems. In the Proceedings of SEKE'99 Eleventh International Conference on Software Engineering and Knowledge Engineering, 1999.

[25] HP Workload manager http://h30081.www3.hp.com/products/wlm/